Student : Esteban Ordenes

Post Graduate Program in Data Science and Business Analytics

PGP-DSBA-UTA-Dec20-A

Travel Package Purchase Prediction

Context

Currently, there are 5 types of packages the company is offering - Basic, Standard, Deluxe, Super Deluxe, King. Looking at the data of the last year, we observed that 18% of the customers purchased the packages.

However, the marketing cost was quite high because customers were contacted at random without looking at the available information.

The company is now planning to launch a new product i.e. Wellness Tourism Package. Wellness Tourism is defined as Travel that allows the traveler to maintain, enhance or kick-start a healthy lifestyle, and support or increase one's sense of well-being.

However, this time company wants to harness the available data of existing and potential customers to make the marketing expenditure more efficient.

Objective

Predict which customer is more likely to purchase the newly introduced travel package.

Data Dictionary

- CustomerID: Unique customer ID
- ProdTaken: Whether the customer has purchased a package or not (0: No, 1: Yes)
- Age: Age of customer
- TypeofContact: How customer was contacted (Company Invited or Self Inquiry)
- CityTier: City tier depends on the development of a city, population, facilities, and living standards. The categories are ordered i.e. Tier 1 > Tier 2 > Tier 3
- Occupation: Occupation of customer
- Gender: Gender of customer
- NumberOfPersonVisiting: Total number of persons planning to take the trip with the customer
- PreferredPropertyStar: Preferred hotel property rating by customer
- MaritalStatus: Marital status of customer
- NumberOfTrips: Average number of trips in a year by customer
- Passport: The customer has a passport or not (0: No, 1: Yes)
- OwnCar: Whether the customers own a car or not (0: No, 1: Yes)

- NumberOfChildrenVisiting: Total number of children with age less than 5 planning to take the trip with the customer
- Designation: Designation of the customer in the current organization
- MonthlyIncome: Gross monthly income of the customer

Customer interaction data:

- PitchSatisfactionScore: Sales pitch satisfaction score
- ProductPitched: Product pitched by the salesperson
- NumberOfFollowups: Total number of follow-ups has been done by the salesperson after the sales pitch
- DurationOfPitch: Duration of the pitch by a salesperson to the customer

Loading libraries

In [163…

```
import warnings
warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data

import pandas as pd
import numpy as np
import scipy.stats as stats

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Library to split data
from sklearn.model_selection import train_test_split

#libraries to help with model building
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree



# Removes the limit from the number of displayed columns and rows.
# This is so I can see the entire dataframe when I  print it
pd.set_option("display.max_columns", None)
# pd.set_option('display.max_rows', None)
pd.set_option("display.max_rows", 200)

# To build linear model for statistical analysis and prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant

# To get diferent metric scores
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_scor

from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import train_test_split

# For pandas profiling
#from pandas_profiling import ProfileReport

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingRegressor, AdaBoostRegressor, StackingRegr
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier



import warnings
warnings.filterwarnings('ignore')
```

Read the dataset

In [3]:
```
Loan = pd.read_csv("Tourism.csv")
```

In [4]:
```
# copying data to another variable to avoid any changes to original data
data = Loan.copy()
```

View the first and last 5 rows of the dataset.

In [5]:
```
data.head()
```

Out[5]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numb
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	Female	
1	200001	0	49.0	Company Invited	1	14.0	Salaried	Male	
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male	
3	200003	0	33.0	Company Invited	1	9.0	Salaried	Female	
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	Male	

In [6]:
```
data.tail()
```

Out[6]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Nu
4883	204883	1	49.0	Self Enquiry	3	9.0	Small Business	Male	
4884	204884	1	28.0	Company Invited	1	31.0	Salaried	Male	
4885	204885	1	52.0	Self Enquiry	3	17.0	Salaried	Female	

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Nu
4886	204886	1	19.0	Self Enquiry	3	16.0	Small Business	Male	
4887	204887	1	36.0	Self Enquiry	1	14.0	Salaried	Male	

Understand the shape of the dataset.

In [7]:
```
data.shape
```

Out[7]:
```
(4888, 20)
```

- The dataset has 4888 rows and 20 columns

Check data types and number of non-null values for each column.

In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   CustomerID               4888 non-null   int64  
 1   ProdTaken                4888 non-null   int64  
 2   Age                      4662 non-null   float64
 3   TypeofContact            4863 non-null   object 
 4   CityTier                 4888 non-null   int64  
 5   DurationOfPitch          4637 non-null   float64
 6   Occupation               4888 non-null   object 
 7   Gender                   4888 non-null   object 
 8   NumberOfPersonVisiting   4888 non-null   int64  
 9   NumberOfFollowups        4843 non-null   float64
 10  ProductPitched           4888 non-null   object 
 11  PreferredPropertyStar    4862 non-null   float64
 12  MaritalStatus            4888 non-null   object 
 13  NumberOfTrips            4748 non-null   float64
 14  Passport                 4888 non-null   int64  
 15  PitchSatisfactionScore   4888 non-null   int64  
 16  OwnCar                   4888 non-null   int64  
 17  NumberOfChildrenVisiting 4822 non-null   float64
 18  Designation              4888 non-null   object 
 19  MonthlyIncome            4655 non-null   float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB
```

- We can see that there are total 20 columns and 4888 number of rows in the dataset.
- All columns' data type is either integer, float or object.
- There is a number of non-null values in some of the columns. We can further confirm this using isna() method.

In [9]:
```
data.isna().sum()
```

Out[9]:
```
CustomerID                    0
ProdTaken                     0
Age                         226
```

```
TypeofContact              25
CityTier                    0
DurationOfPitch           251
Occupation                  0
Gender                      0
NumberOfPersonVisiting      0
NumberOfFollowups          45
ProductPitched              0
PreferredPropertyStar      26
MaritalStatus               0
NumberOfTrips             140
Passport                    0
PitchSatisfactionScore      0
OwnCar                      0
NumberOfChildrenVisiting   66
Designation                 0
MonthlyIncome             233
dtype: int64
```

Summary of the dataset

In [10]:
```
# Summary of continuous columns
data[[ 'DurationOfPitch' , 'NumberOfTrips' , 'MonthlyIncome']].describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
DurationOfPitch	4637.0	15.490835	8.519643	5.0	9.0	13.0	20.0	127.0
NumberOfTrips	4748.0	3.236521	1.849019	1.0	2.0	3.0	4.0	22.0
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0	20346.0	22347.0	25571.0	98678.0

- `DurationOfPitch` has missing values, mean and median 15.5 and 13 respectively.
- `NumberOfFollowups` has missing values, mean and median is 3.7 and 4 respectively.
- `NumberOfTrips` has missing values, mean and median is 3.23 and 3.0 respectively.
- Mean and median value for `NumberOfChildrenVisiting` is 1.18 and 1.0 respectively,
- `MonthlyIncome` has missing values, mean and median is 23619.85 and 22347.0 respectively.

Converting the data type of categorical features to 'category'

In [11]:
```
cat_cols = ['Age', 'CityTier'
            , 'PreferredPropertyStar'
            , 'Passport'
            , 'PitchSatisfactionScore'
            , 'OwnCar'
            , 'TypeofContact'
            , 'Occupation'
            , 'Gender'
            , 'ProductPitched'
            , 'MaritalStatus'
            , 'Designation']
data[cat_cols] = data[cat_cols].astype('category')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
```

```
 0   CustomerID               4888 non-null   int64
 1   ProdTaken                4888 non-null   int64
 2   Age                      4662 non-null   category
 3   TypeofContact            4863 non-null   category
 4   CityTier                 4888 non-null   category
 5   DurationOfPitch          4637 non-null   float64
 6   Occupation               4888 non-null   category
 7   Gender                   4888 non-null   category
 8   NumberOfPersonVisiting   4888 non-null   int64
 9   NumberOfFollowups        4843 non-null   float64
 10  ProductPitched           4888 non-null   category
 11  PreferredPropertyStar    4862 non-null   category
 12  MaritalStatus            4888 non-null   category
 13  NumberOfTrips            4748 non-null   float64
 14  Passport                 4888 non-null   category
 15  PitchSatisfactionScore   4888 non-null   category
 16  OwnCar                   4888 non-null   category
 17  NumberOfChildrenVisiting 4822 non-null   float64
 18  Designation              4888 non-null   category
 19  MonthlyIncome            4655 non-null   float64
dtypes: category(12), float64(5), int64(3)
memory usage: 366.0 KB
```

In [12]:
```
# Summary of categorical columns
data[[ 'ProdTaken'
      , 'TypeofContact'
      , 'CityTier'
      , 'Occupation'
      , 'Gender'
      , 'PreferredPropertyStar'
      , 'ProductPitched'
      , 'NumberOfPersonVisiting'
      , 'NumberOfChildrenVisiting'
      , 'NumberOfFollowups'
      , 'MaritalStatus'
      , 'Designation'
      ,  'Passport'
      , 'PitchSatisfactionScore'
      , 'OwnCar'
      ]].describe(include='category').T
```

Out[12]:

	count	unique	top	freq
TypeofContact	4863	2	Self Enquiry	3444
CityTier	4888	3	1	3190
Occupation	4888	4	Salaried	2368
Gender	4888	3	Male	2916
PreferredPropertyStar	4862	3	3	2993
ProductPitched	4888	5	Basic	1842
MaritalStatus	4888	4	Married	2340
Designation	4888	5	Executive	1842
Passport	4888	2	0	3466
PitchSatisfactionScore	4888	5	3	1478
OwnCar	4888	2	1	3032

In [13]:

```python
# inspect discrete columns
print('\n\nAge')
print(data.Age.value_counts())
print('\n\nTypeofContact')
print(data.TypeofContact.value_counts())
print('\n\nCityTier')
print(data.CityTier.value_counts())
print('\n\nOccupation')
print(data.Occupation.value_counts())
print('\n\nGender')
print(data.Gender.value_counts())
print('\n\nPreferredPropertyStar')
print(data.PreferredPropertyStar.value_counts())
print('\n\nProductPitched')
print(data.ProductPitched.value_counts())
print('\n\nMaritalStatus')
print(data.MaritalStatus.value_counts())
print('\n\nNumberOfPersonVisiting')
print(data.NumberOfPersonVisiting.value_counts())
print('\n\nNumberOfChildrenVisiting')
print(data.NumberOfChildrenVisiting.value_counts())
print('\n\nNumberOfFollowups')
print(data.NumberOfFollowups.value_counts())
print('\n\nDesignation')
print(data.Designation.value_counts())
print('\n\nPassport')
print(data.Passport.value_counts())
print('\n\nPitchSatisfactionScore')
print(data.PitchSatisfactionScore.value_counts())
print('\n\nOwnCar')
print(data.OwnCar.value_counts())
```

```
Age
35.0    237
36.0    231
34.0    211
31.0    203
30.0    199
32.0    197
33.0    189
37.0    185
29.0    178
38.0    176
41.0    155
39.0    150
28.0    147
40.0    146
42.0    142
27.0    138
43.0    130
46.0    121
45.0    116
26.0    106
44.0    105
51.0     90
47.0     88
50.0     86
25.0     74
52.0     68
53.0     66
48.0     65
```

```
49.0     65
55.0     64
54.0     61
56.0     58
24.0     56
23.0     46
22.0     46
59.0     44
21.0     41
20.0     38
19.0     32
58.0     31
57.0     29
60.0     29
18.0     14
61.0      9
Name: Age, dtype: int64


TypeofContact
Self Enquiry       3444
Company Invited    1419
Name: TypeofContact, dtype: int64


CityTier
1    3190
3    1500
2     198
Name: CityTier, dtype: int64


Occupation
Salaried          2368
Small Business    2084
Large Business     434
Free Lancer          2
Name: Occupation, dtype: int64


Gender
Male       2916
Female     1817
Fe Male     155
Name: Gender, dtype: int64


PreferredPropertyStar
3.0    2993
5.0     956
4.0     913
Name: PreferredPropertyStar, dtype: int64


ProductPitched
Basic           1842
Deluxe          1732
Standard         742
Super Deluxe     342
King             230
Name: ProductPitched, dtype: int64


MaritalStatus
Married      2340
```

```
Divorced      950
Single        916
Unmarried     682
Name: MaritalStatus, dtype: int64


NumberOfPersonVisiting
3    2402
2    1418
4    1026
1      39
5       3
Name: NumberOfPersonVisiting, dtype: int64


NumberOfChildrenVisiting
1.0    2080
2.0    1335
0.0    1082
3.0     325
Name: NumberOfChildrenVisiting, dtype: int64


NumberOfFollowups
4.0    2068
3.0    1466
5.0     768
2.0     229
1.0     176
6.0     136
Name: NumberOfFollowups, dtype: int64


Designation
Executive         1842
Manager           1732
Senior Manager     742
AVP                342
VP                 230
Name: Designation, dtype: int64


Passport
0    3466
1    1422
Name: Passport, dtype: int64


PitchSatisfactionScore
3    1478
5     970
1     942
4     912
2     586
Name: PitchSatisfactionScore, dtype: int64


OwnCar
1    3032
0    1856
Name: OwnCar, dtype: int64
```

- Gender seems to have rows with type `Fe Male` .this will be fixed

In [14]:
```python
def fixGenderValues(gender):
    if gender == 'Fe Male' :
        return 'Female'
    else:
        return gender
```

In [15]:
```python
data['Gender'] = data['Gender'].apply(fixGenderValues)
```

In [16]:
```python
# check for unique values for each column
data.nunique()
```

Out[16]:
```
CustomerID                  4888
ProdTaken                      2
Age                           44
TypeofContact                  2
CityTier                       3
DurationOfPitch               34
Occupation                     4
Gender                         2
NumberOfPersonVisiting         5
NumberOfFollowups              6
ProductPitched                 5
PreferredPropertyStar          3
MaritalStatus                  4
NumberOfTrips                 12
Passport                       2
PitchSatisfactionScore         5
OwnCar                         2
NumberOfChildrenVisiting       4
Designation                    5
MonthlyIncome               2475
dtype: int64
```

- We can drop 'CustomerID' column as it is an ID variable and will not add value to the model.

In [17]:
```python
#Dropping CustomerID columns from the dataframe
data.drop(columns=['CustomerID'], inplace=True)
```

Lets Evaluate the Dependant Variable - ProdTaken

In [18]:
```python
data['ProdTaken'].value_counts()
```

Out[18]:
```
0    3968
1     920
Name: ProdTaken, dtype: int64
```

- 18% of the customers purchased the packages

EDA

Univariate analysis

In [19]:
```python
def histogram_boxplot(feature , figsize=(15,10) , bins=None):
    """ Histogram and Boxplot combined
    feature: 1-d feature array
    figsize: size of figg.default (15,10)
```

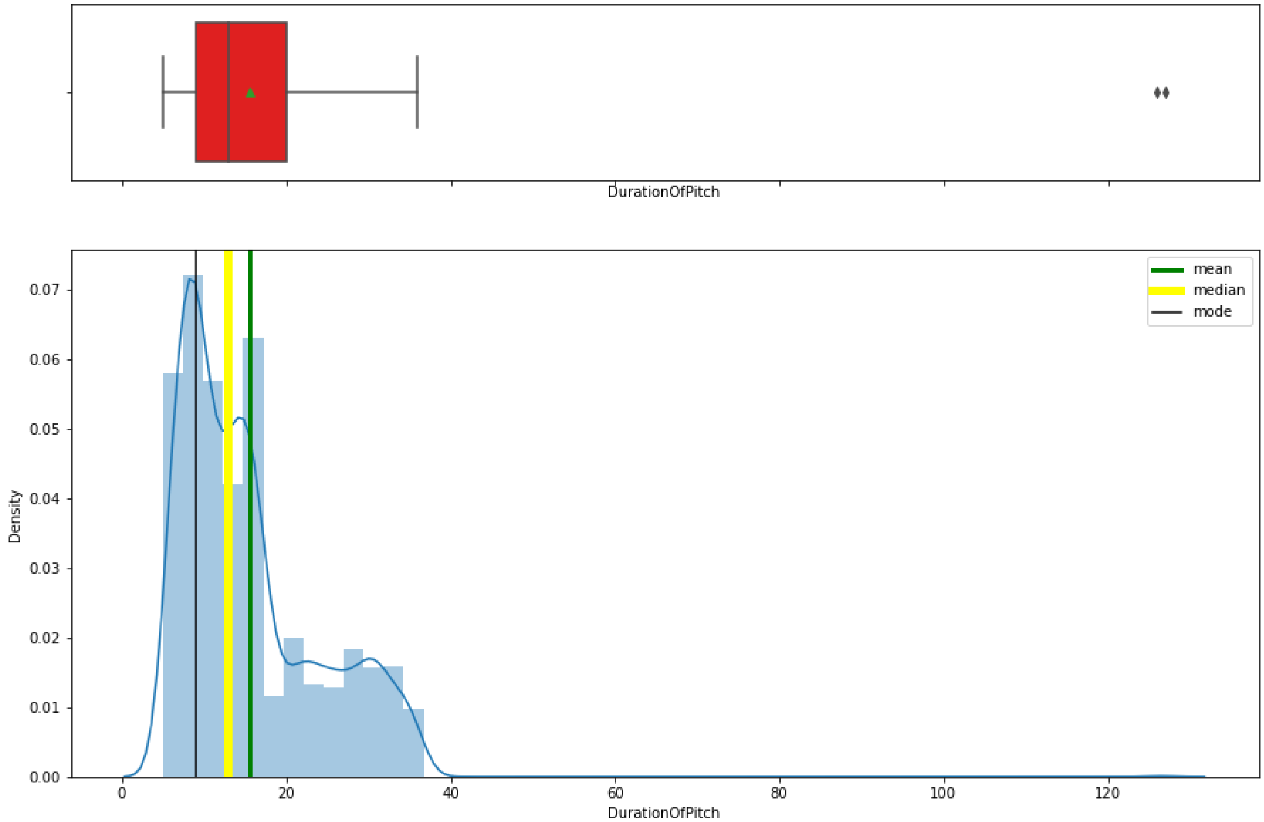
```
    bins: number of bins.default None/auto
    """
    mean = feature.mean()
    median = feature.median()
    mode = feature.mode()

    f2, (ax_box2 , ax_hist2) = plt.subplots(nrows = 2, # num of rows of the subplot. gr
                                            sharex = True, # x-axis will be shared amon
                                            gridspec_kw = { "height_ratios": (.25 , .75
                                            figsize = figsize
                                           ) # create the 2 subplots

    sns.boxplot(feature , ax = ax_box2 , showmeans = True , color = 'red') # boxplot wi
    if bins:
        sns.distplot(feature , kde = True , ax = ax_hist2, bins = bins)
    else:
        sns.distplot( feature , kde = True , ax = ax_hist2 )
    ax_hist2.axvline( mean , color = 'green' , linestyle='-' , linewidth = 3 , label =
    ax_hist2.axvline( median , color = 'yellow' , linestyle='-' , linewidth = 6 , label
    ax_hist2.axvline( mode[0] , color = 'black' , linestyle='-' , label = 'mode' ) # ad
    ax_hist2.legend()

    print( 'Mean:'+ str( mean ) )
    print( 'Median:'+ str( median ) )
    print( 'Mode:'+ str( mode[0] ) )
```

In [20]:
```
def bar_count_pct( feature , figsize=(10,7) ):
    """
    feature : 1-d categorical feature array
    """
    mode = feature.mode()
    freq = feature.value_counts().max()

    #if isinstance(feature , int):
    #    cnt = feature.unique()
    #else:
    #    cnt  = feature.unique().value_counts().sum()

    plt.figure(figsize=figsize)

    ax = sns.countplot(feature)

    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format( 100 * p.get_height() / total ) # percentage of e
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
        y = p.get_y() + p.get_height() # height of the plot
        ax.annotate( percentage , (x,y), size = 12) # annotate the percentage

    print( 'Top:'+ str( mode[0] ) )
    print( 'Freq:'+ str( freq ) )
```

Observation on DurationOfPitch

In [21]:
```
histogram_boxplot(data.DurationOfPitch)
```

```
Mean:15.490834591330602
Median:13.0
Mode:9.0
```

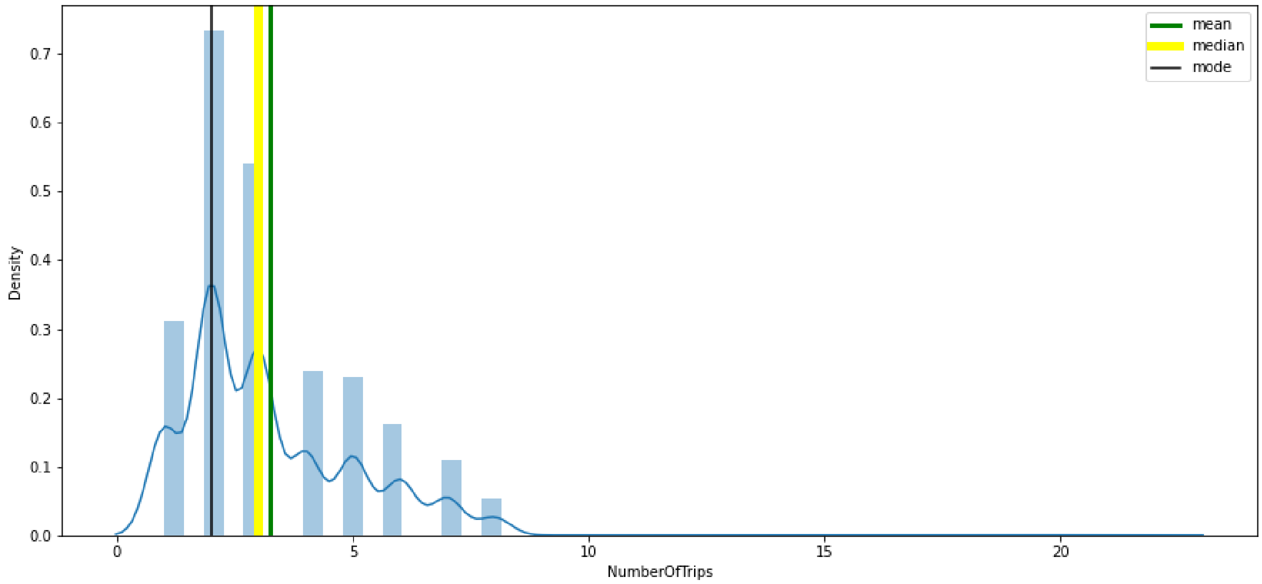


- DurationOfPitch feature is right-skewed.
- There are outliers to the right of the curve which may explain its skewness.

Observation on NumberOfTrips

In [22]:
```
histogram_boxplot(data.NumberOfTrips)
```

```
Mean:3.236520640269587
Median:3.0
Mode:2.0
```

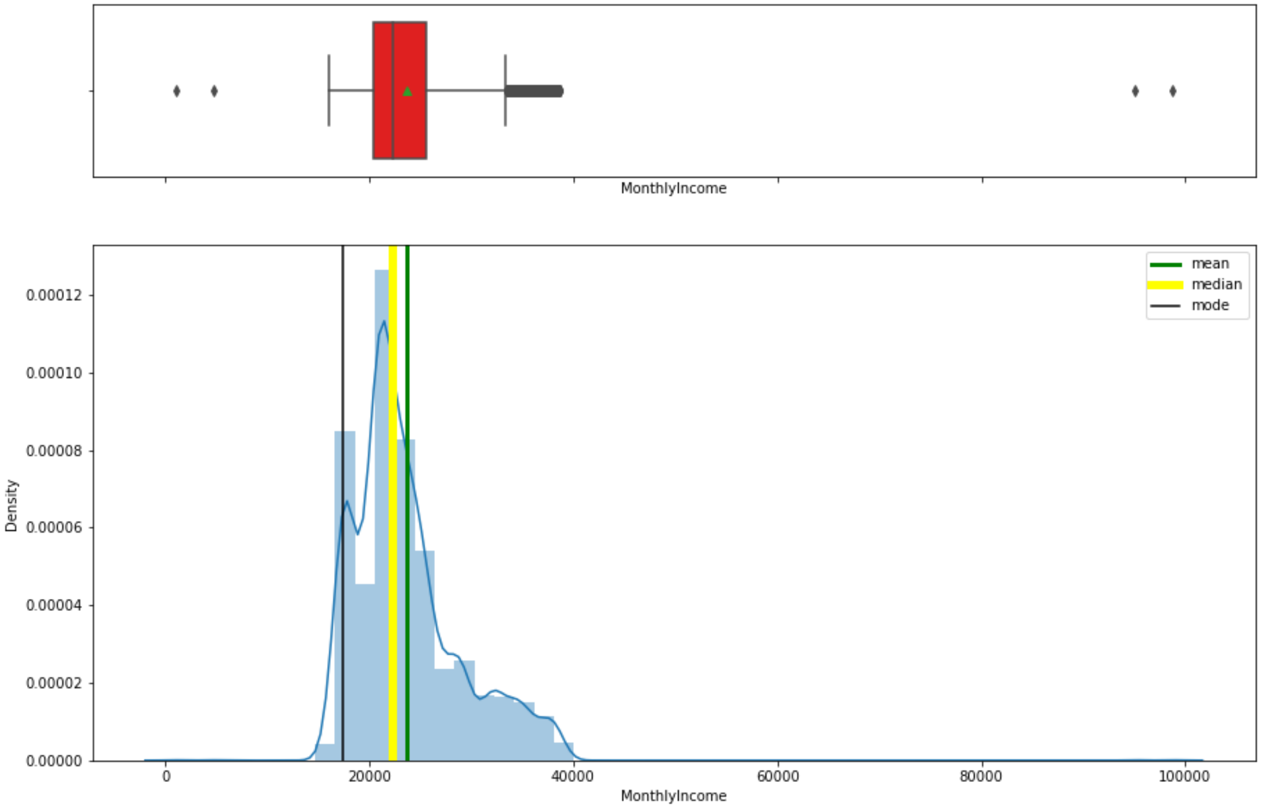


- NumberOfTrips feature is right-skewed.
- There are outliers to the right of the curve which may explain its skewness.

Observation on MonthlyIncome

In [23]:
```
histogram_boxplot(data.MonthlyIncome)
```

```
Mean:23619.85349087003
Median:22347.0
Mode:17342.0
```

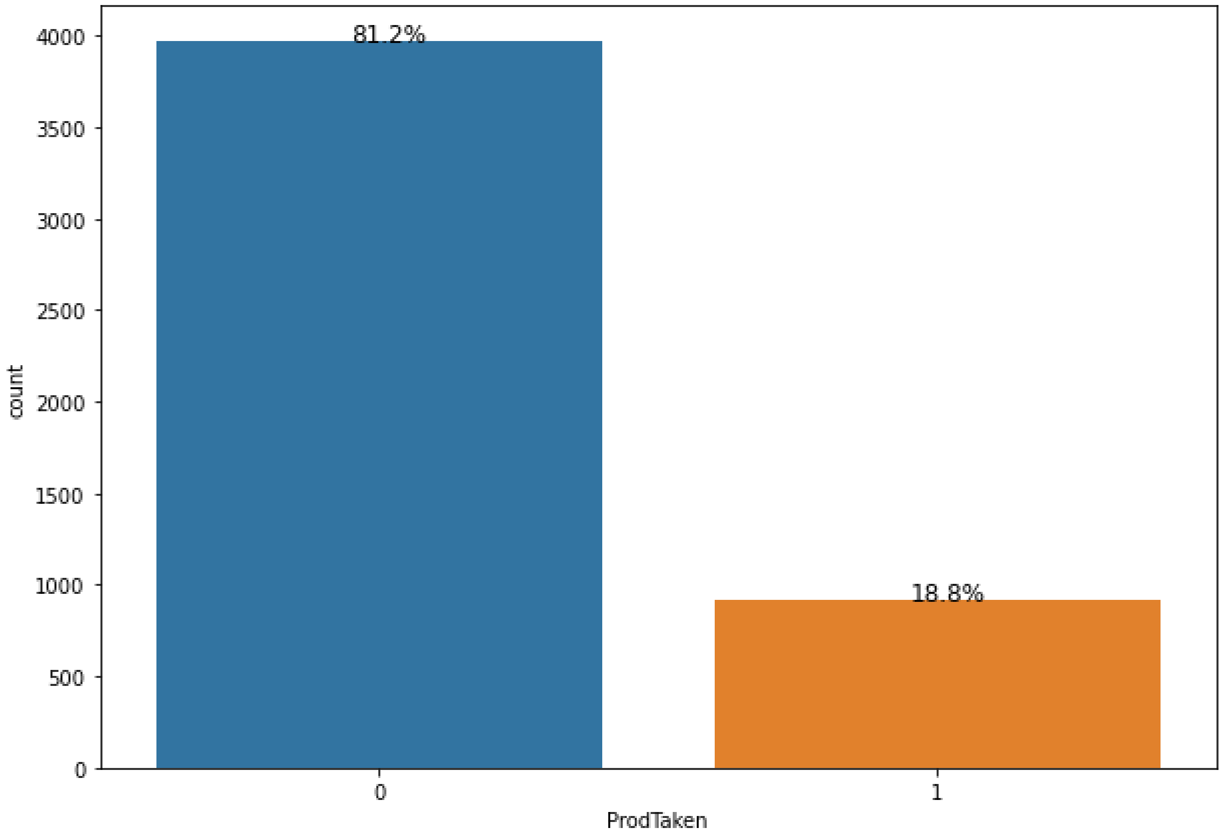


- MonthlyIncome feature is right-skewed.
- There are outliers to the right of the curve which may explain its skewness.

Observations on ProdTaken (Dependant Variable)

In [24]:
```
print('ProdTaken\n' , data['ProdTaken'].value_counts(normalize=True) , '\n')
bar_count_pct(data.ProdTaken)
```

```
ProdTaken
 0    0.811784
1    0.188216
Name: ProdTaken, dtype: float64

Top:0
Freq:3968
```



- Mode frequent ProdTaken is False(0) with 82%.
- Only 18% of have ProdTaken is True(1).
- There are 2 (True/False or 0/1) unique values.

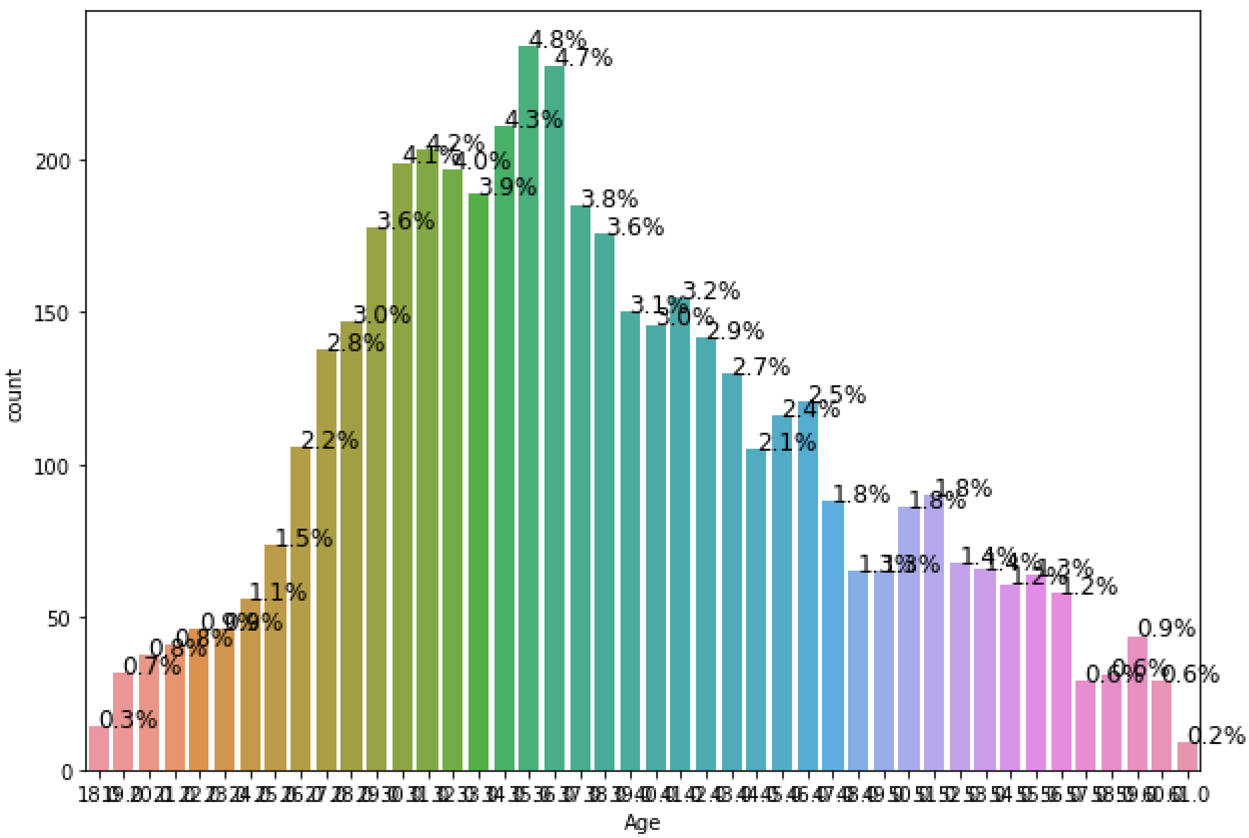
Observations on Age

In [25]:

```
print('Age\n' , data['Age'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Age)
```

```
Age
 35.0    0.050837
36.0    0.049550
34.0    0.045260
31.0    0.043544
30.0    0.042686
32.0    0.042257
33.0    0.040541
37.0    0.039683
29.0    0.038181
38.0    0.037752
41.0    0.033248
39.0    0.032175
28.0    0.031532
40.0    0.031317
42.0    0.030459
27.0    0.029601
43.0    0.027885
46.0    0.025955
45.0    0.024882
26.0    0.022737
44.0    0.022523
51.0    0.019305
47.0    0.018876
```

```
50.0    0.018447
25.0    0.015873
52.0    0.014586
53.0    0.014157
48.0    0.013943
49.0    0.013943
55.0    0.013728
54.0    0.013085
56.0    0.012441
24.0    0.012012
23.0    0.009867
22.0    0.009867
59.0    0.009438
21.0    0.008795
20.0    0.008151
19.0    0.006864
58.0    0.006650
57.0    0.006221
60.0    0.006221
18.0    0.003003
61.0    0.001931
Name: Age, dtype: float64

Top:35.0
Freq:237
```



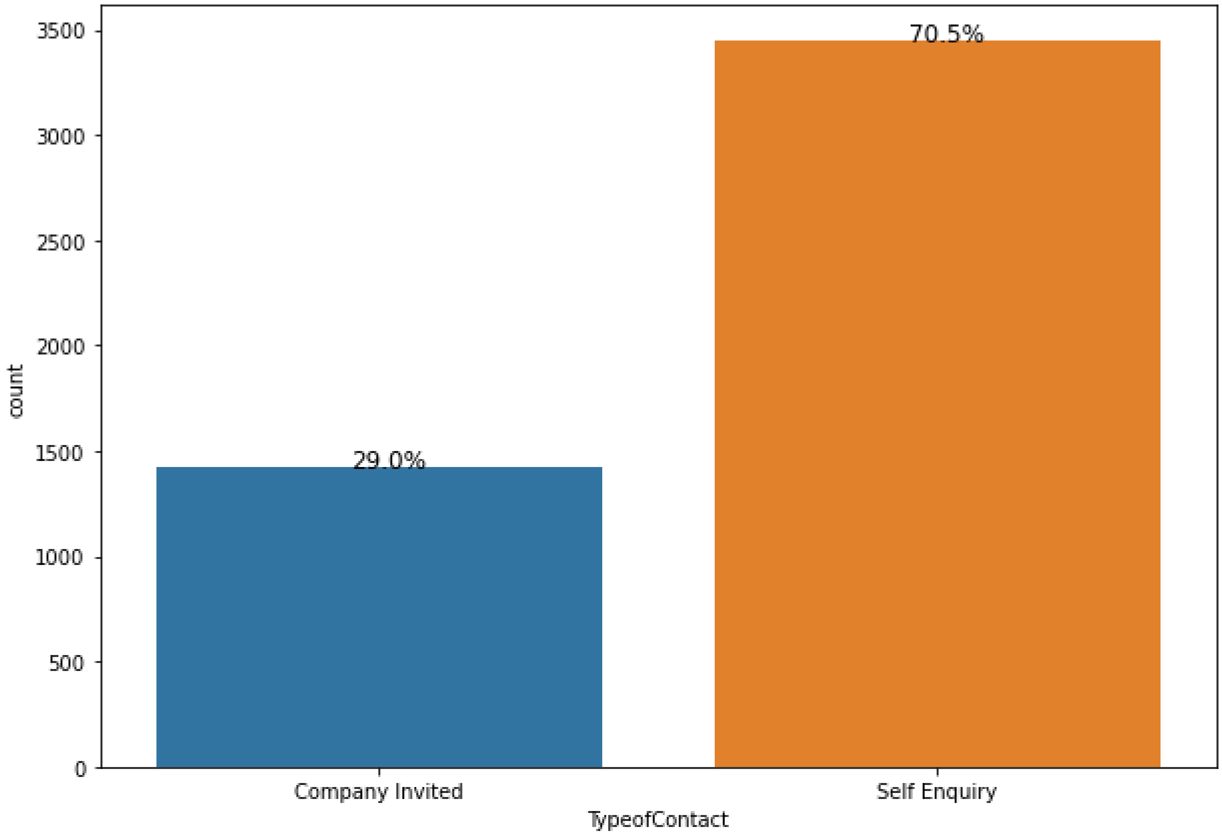
- Most common Age is 35, it has a frequency of 237.

Observations on TypeofContact

In [26]:
```python
print('TypeofContact\n' , data['TypeofContact'].value_counts(normalize=True) , '\n')
bar_count_pct(data.TypeofContact)
```

```
TypeofContact
 Self Enquiry      0.708205
```

```
Company Invited    0.291795
Name: TypeofContact, dtype: float64

Top:Self Enquiry
Freq:3444
```



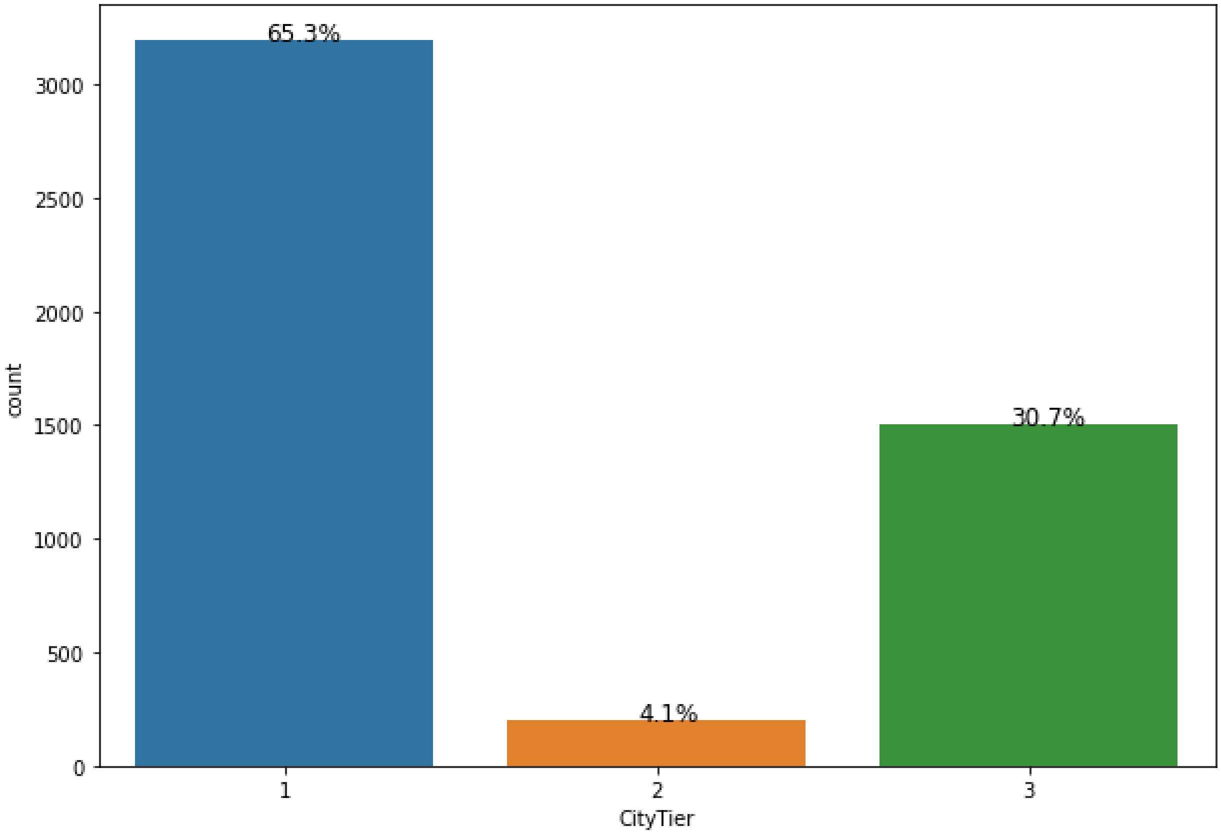
- Mode frequent TypeofContact is 'Self Enquiry' with 71%.
- Only 29% of have 'Company Invited'.
- There are 2 unique values.

Observations on CityTier

In [27]:

```
print('CityTier\n' , data['CityTier'].value_counts(normalize=True) , '\n')
bar_count_pct(data.CityTier)
```

```
CityTier
 1    0.652619
3    0.306874
2    0.040507
Name: CityTier, dtype: float64

Top:1
Freq:3190
```

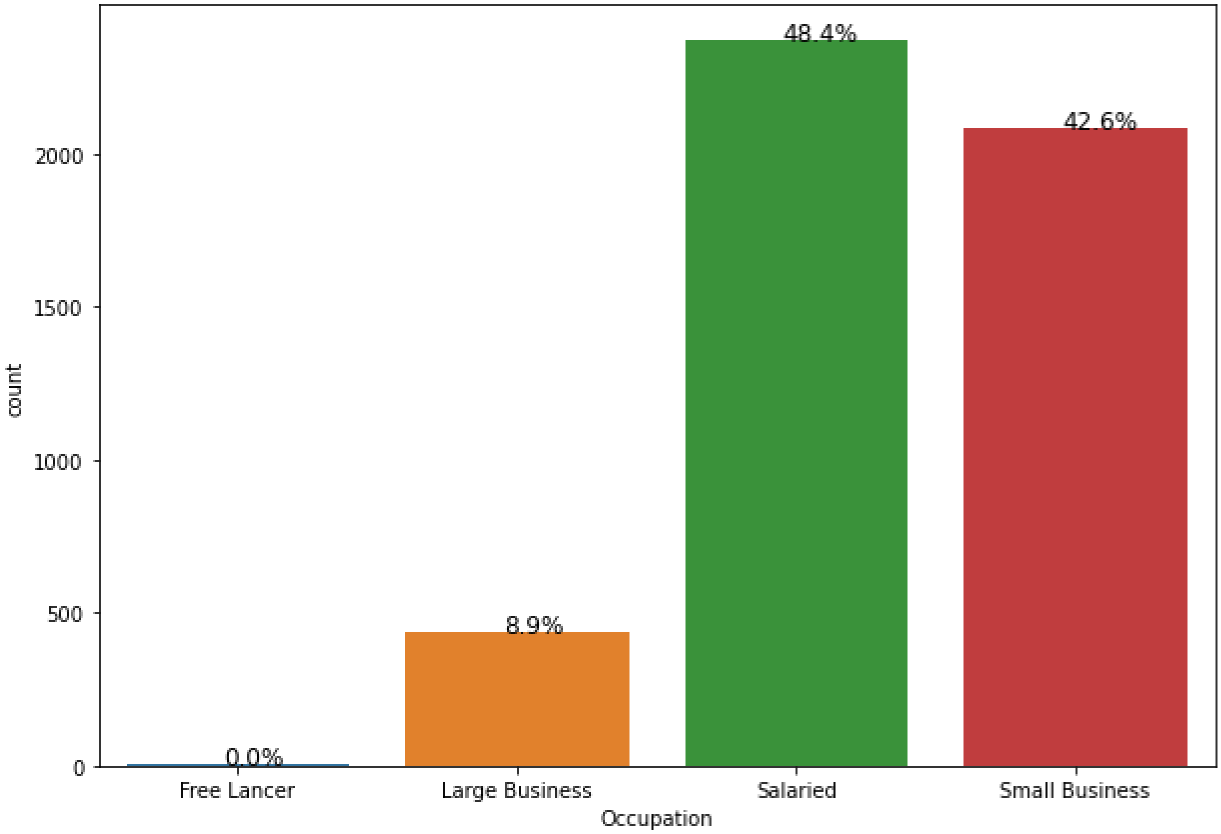


- Mode frequent CityTier is Tier-1 with 65%.
- Only 4.1% of are Tier-2.
- There are 3 unique values.

Observations on Occupation

In [28]:
```
print('Occupation\n' , data['Occupation'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Occupation)
```

```
Occupation
 Salaried          0.484452
Small Business    0.426350
Large Business    0.088789
Free Lancer       0.000409
Name: Occupation, dtype: float64

Top:Salaried
Freq:2368
```

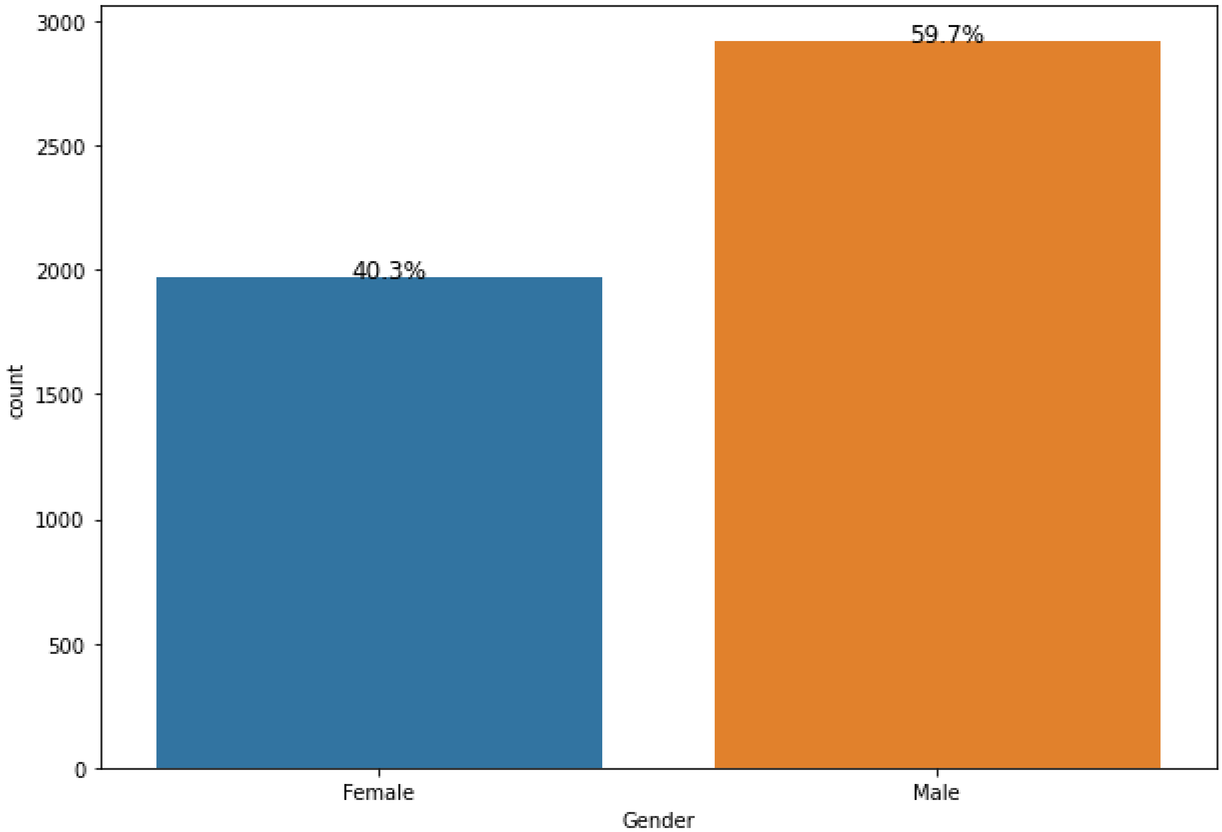


- Most frequent Occupation is Salaried with 48%.
- very few 0% are Free Lancer
- There are 4 unique values.

Observations on Gender

In [29]:
```
print('Gender\n' , data['Gender'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Gender)
```

```
Gender
 Male      0.596563
Female    0.403437
Name: Gender, dtype: float64

Top:Male
Freq:2916
```

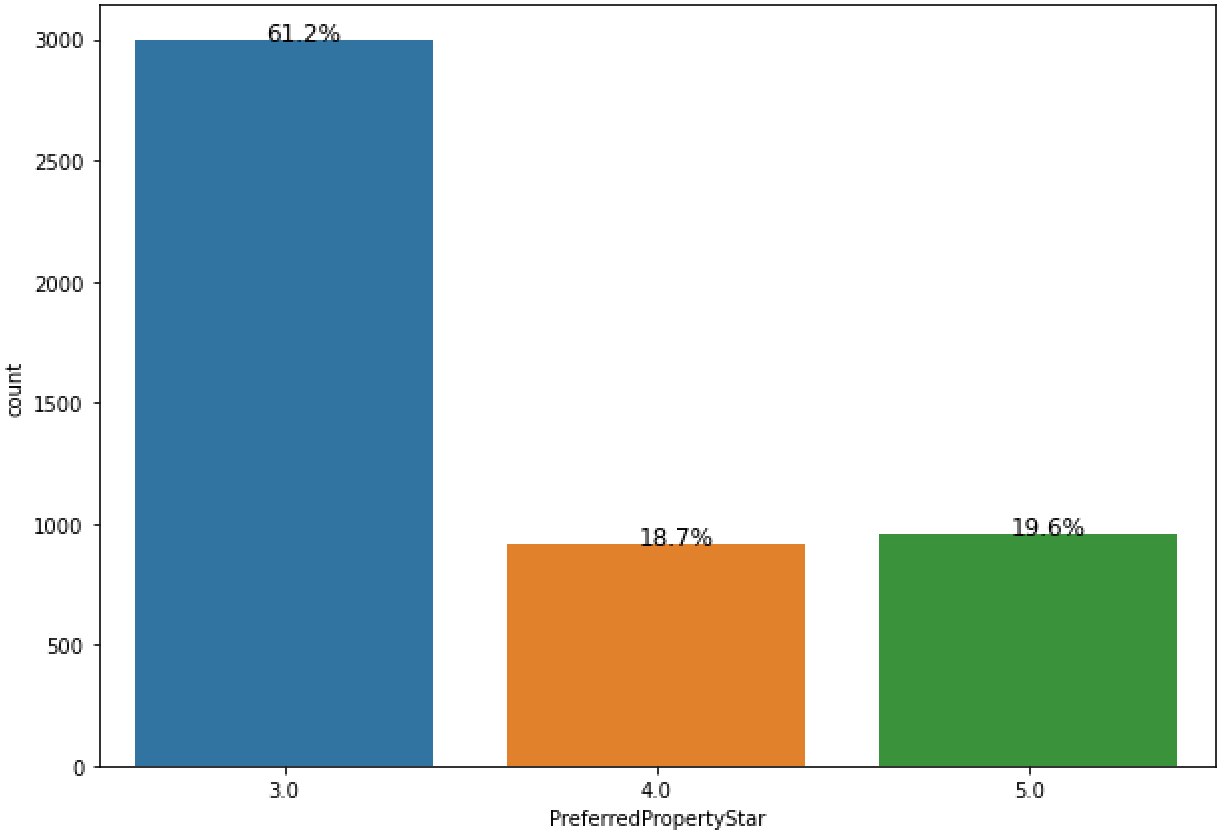


- Most frequent Gender is Male with 60%.
- Female is 40%.
- There are 2 unique values.

Observations on PreferredPropertyStar

In [30]:
```
print('PreferredPropertyStar\n' , data['PreferredPropertyStar'].value_counts(normalize=
bar_count_pct(data.PreferredPropertyStar)
```

```
PreferredPropertyStar
 3.0    0.615590
5.0    0.196627
4.0    0.187783
Name: PreferredPropertyStar, dtype: float64

Top:3.0
Freq:2993
```

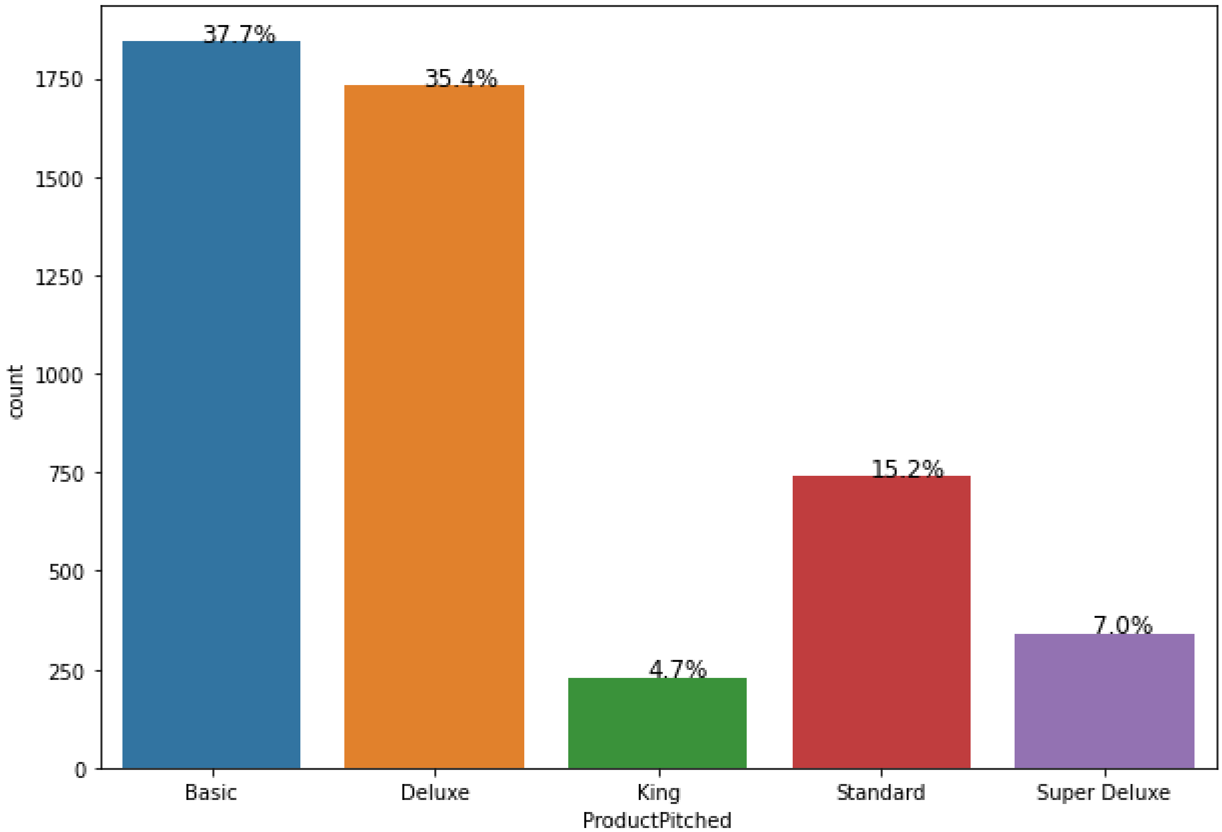


- Most frequent PreferredPropertyStar is 3 with 61%.
- 4 and 5 stars are pretty much even with 18% and 19% respectively.
- There are 3 unique values.

Observations on ProductPitched

In [31]:
```
print('ProductPitched\n' , data['ProductPitched'].value_counts(normalize=True) , '\n')
bar_count_pct(data.ProductPitched)
```

```
ProductPitched
 Basic           0.376841
Deluxe          0.354337
Standard        0.151800
Super Deluxe    0.069967
King            0.047054
Name: ProductPitched, dtype: float64

Top:Basic
Freq:1842
```

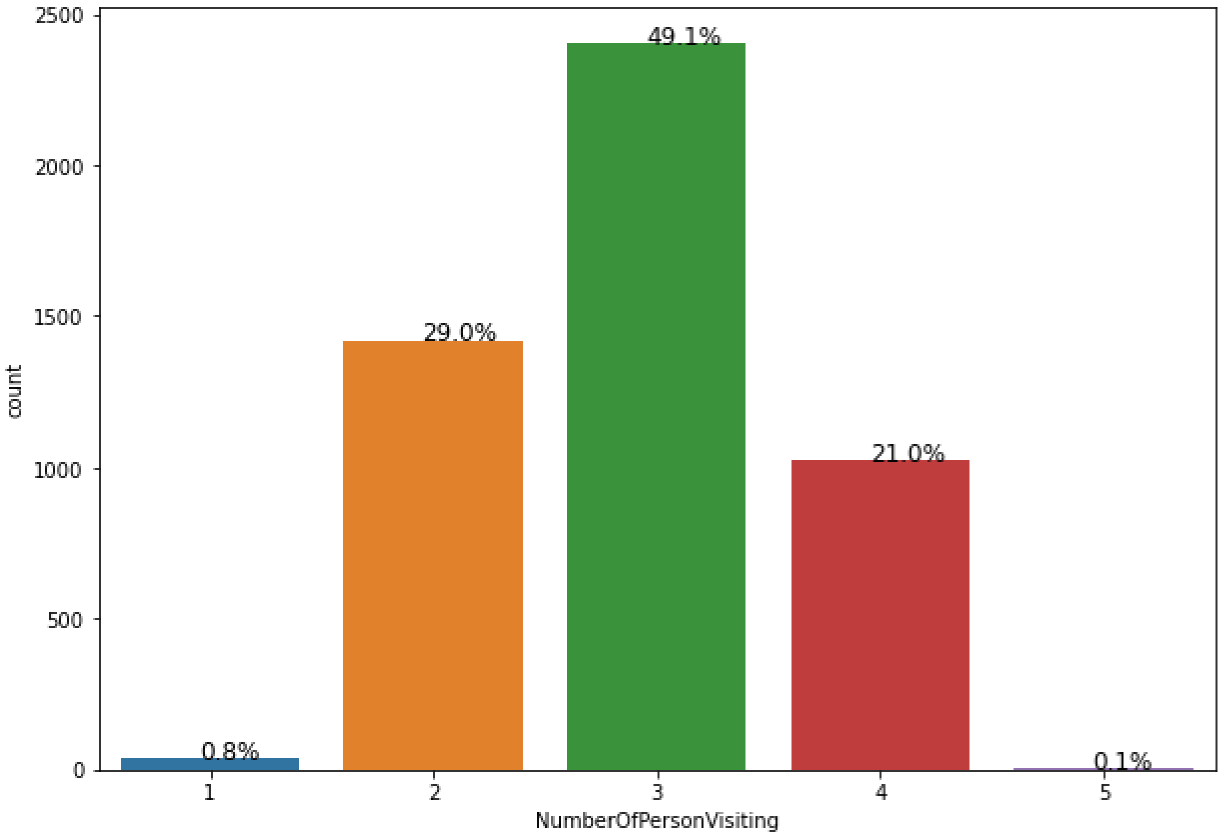


- Most frequent ProductPitched is Basic with 38%.
- King product is the lowest with 4.7%.
- There are 5 unique values.

Observations on NumberOfPersonVisiting

In [32]:
```
print('NumberOfPersonVisiting\n' , data['NumberOfPersonVisiting'].value_counts(normaliz
bar_count_pct(data.NumberOfPersonVisiting)
```

```
NumberOfPersonVisiting
 3    0.491408
2    0.290098
4    0.209902
1    0.007979
5    0.000614
Name: NumberOfPersonVisiting, dtype: float64

Top:3
Freq:2402
```



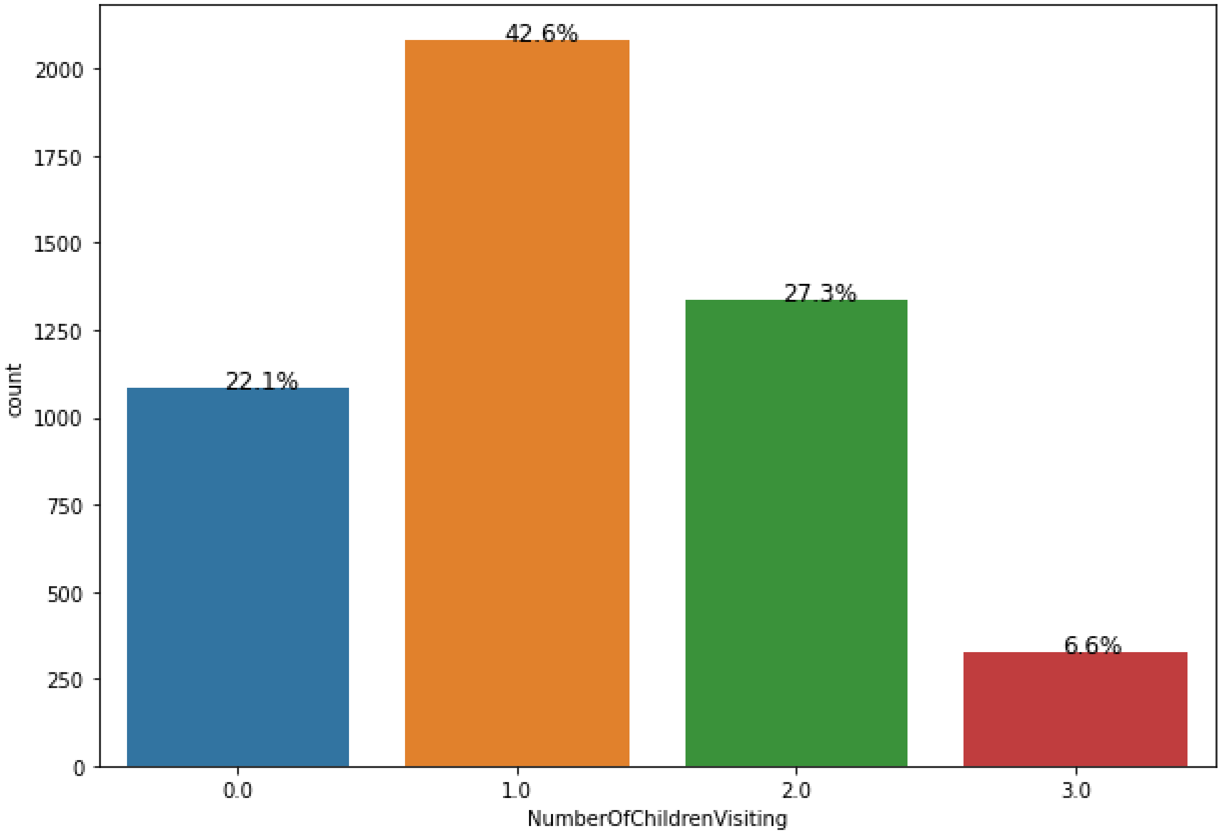
- Most frequent NumberOfPersonVisiting is 3 with 49%.
- 5 is the lowest with 0.1%.
- There are 5 unique values.

Observations on NumberOfChildrenVisiting

In [33]:

```
print('NumberOfChildrenVisiting\n' , data['NumberOfChildrenVisiting'].value_counts(norm
bar_count_pct(data.NumberOfChildrenVisiting)
```

```
NumberOfChildrenVisiting
 1.0    0.431356
2.0    0.276856
0.0    0.224388
3.0    0.067399
Name: NumberOfChildrenVisiting, dtype: float64

Top:1.0
Freq:2080
```

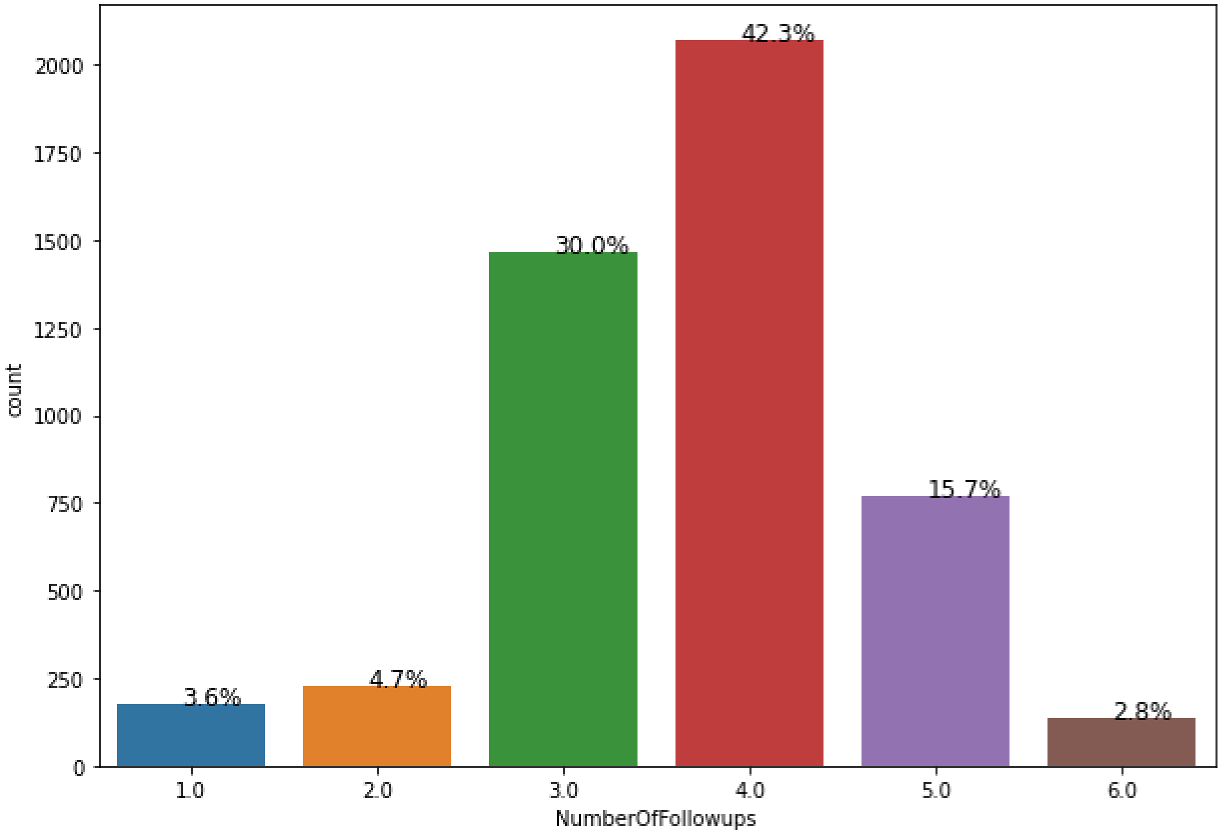


- Most frequent NumberOfChildrenVisiting is 1 with 42%.
- 3 is the lowest with 6.6%.
- There are 4 unique values.

Observations on NumberOfFollowups

In [34]:
```
print('NumberOfFollowups\n' , data['NumberOfFollowups'].value_counts(normalize=True) ,
bar_count_pct(data.NumberOfFollowups)
```

```
NumberOfFollowups
 4.0    0.427008
3.0    0.302705
5.0    0.158579
2.0    0.047285
1.0    0.036341
6.0    0.028082
Name: NumberOfFollowups, dtype: float64

Top:4.0
Freq:2068
```

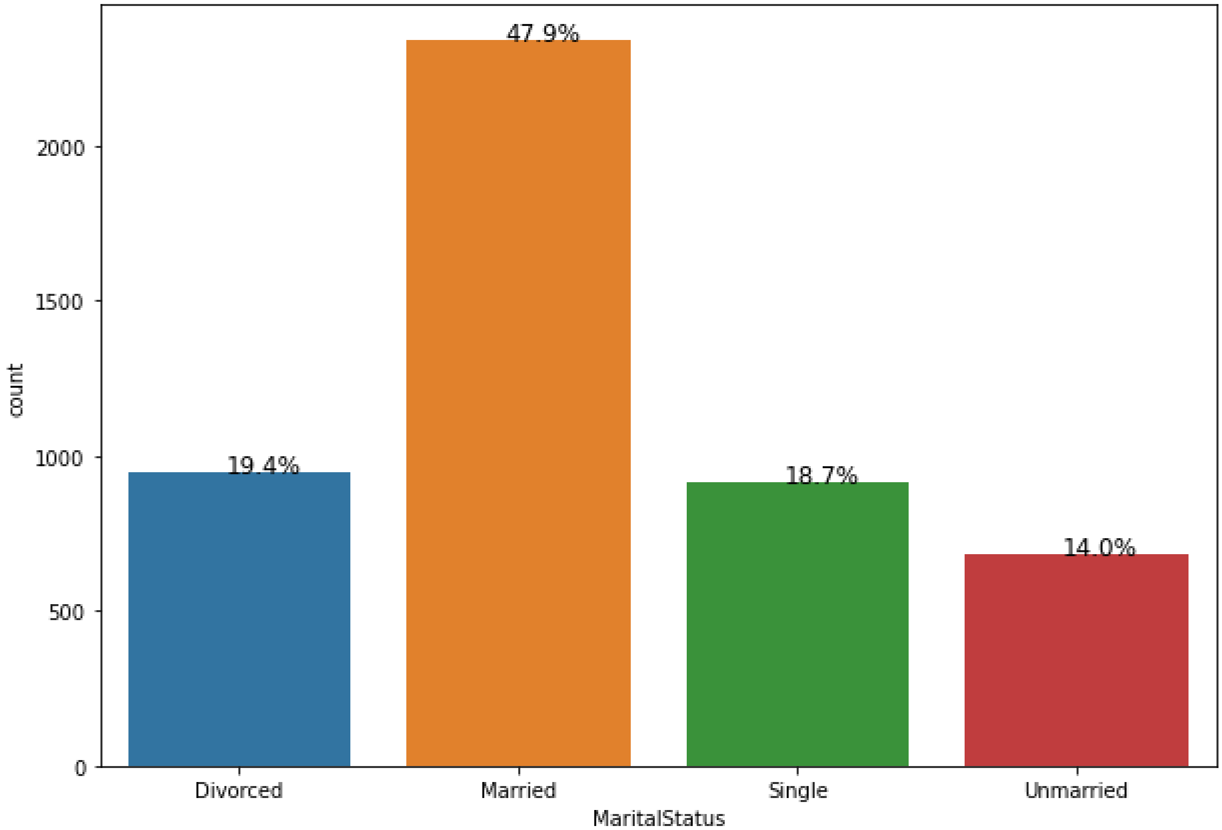


- Most frequent NumberOfFollowups is 4 with 42%.
- 6 is the lowest with 2.8%.
- There are 6 unique values.

Observations on MaritalStatus

In [35]:
```
print('MaritalStatus\n' , data['MaritalStatus'].value_counts(normalize=True) , '\n')
bar_count_pct(data.MaritalStatus)
```

```
MaritalStatus
 Married      0.478723
Divorced     0.194354
Single       0.187398
Unmarried    0.139525
Name: MaritalStatus, dtype: float64

Top:Married
Freq:2340
```

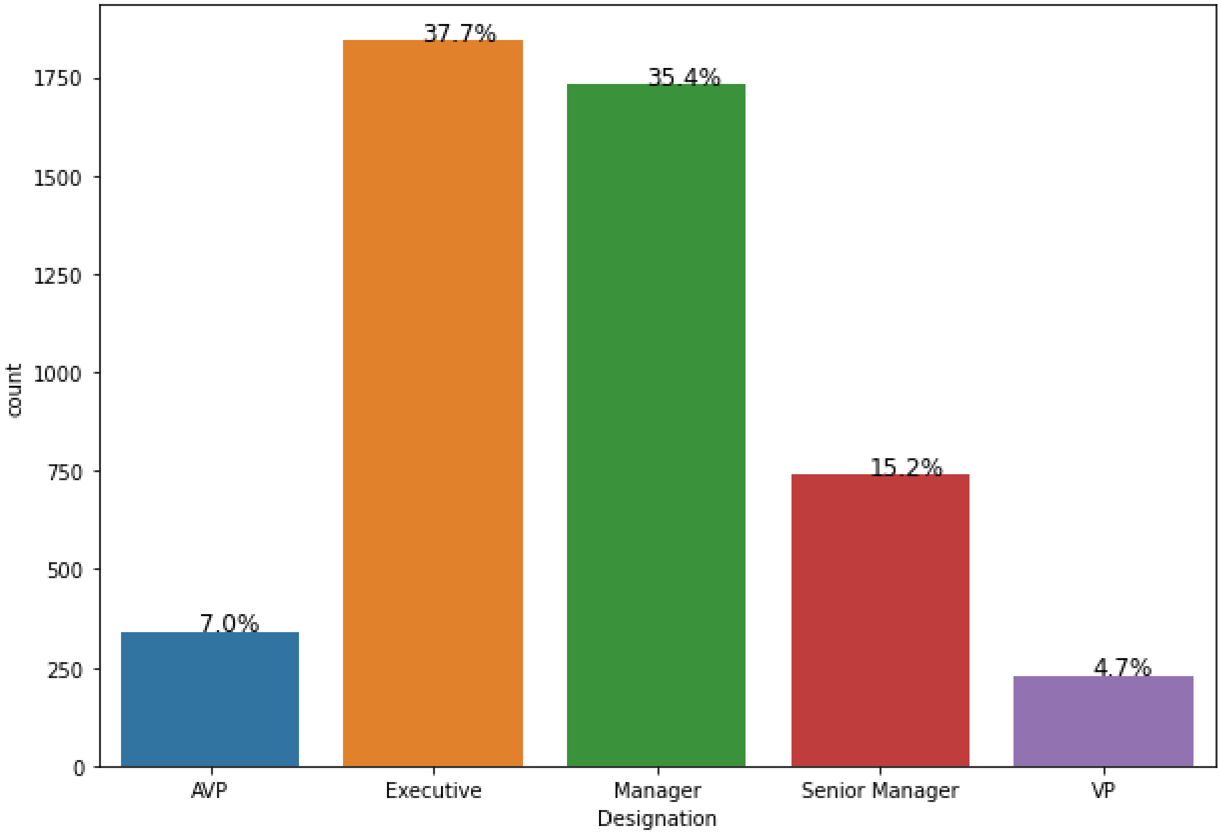


- Most frequent MaritalStatus is Married with 48%.
- Unmarried is the lowest with 14%.
- There are 4 unique values.

Observations on Designation

In [36]:
```
print('Designation\n' , data['Designation'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Designation)
```

```
Designation
 Executive         0.376841
Manager           0.354337
Senior Manager    0.151800
AVP               0.069967
VP                0.047054
Name: Designation, dtype: float64

Top:Executive
Freq:1842
```

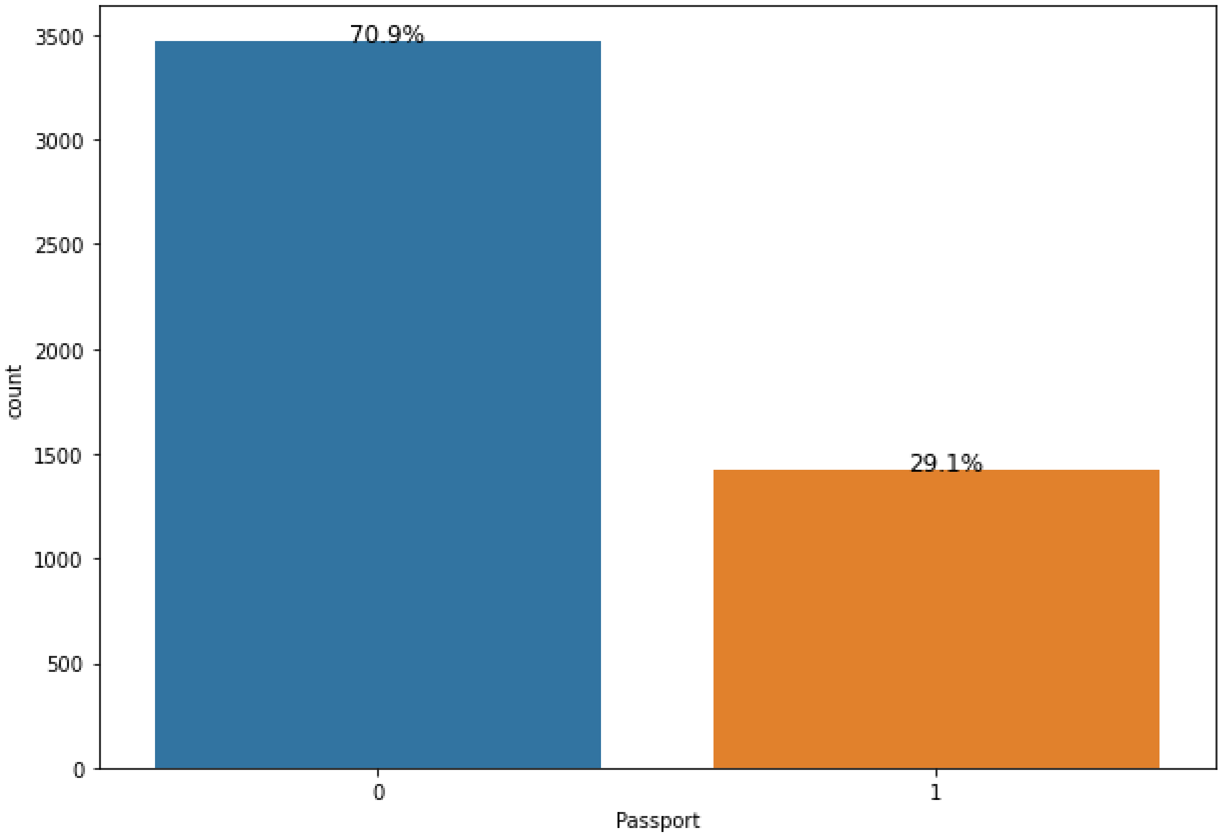


- Most frequent Designation is Executive with 37%.
- VP is the lowest with 4.7%.
- There are 5 unique values.

Observations on Passport

In [37]:
```
print('Passport\n' , data['Passport'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Passport)
```

```
Passport
 0    0.709083
1    0.290917
Name: Passport, dtype: float64

Top:0
Freq:3466
```

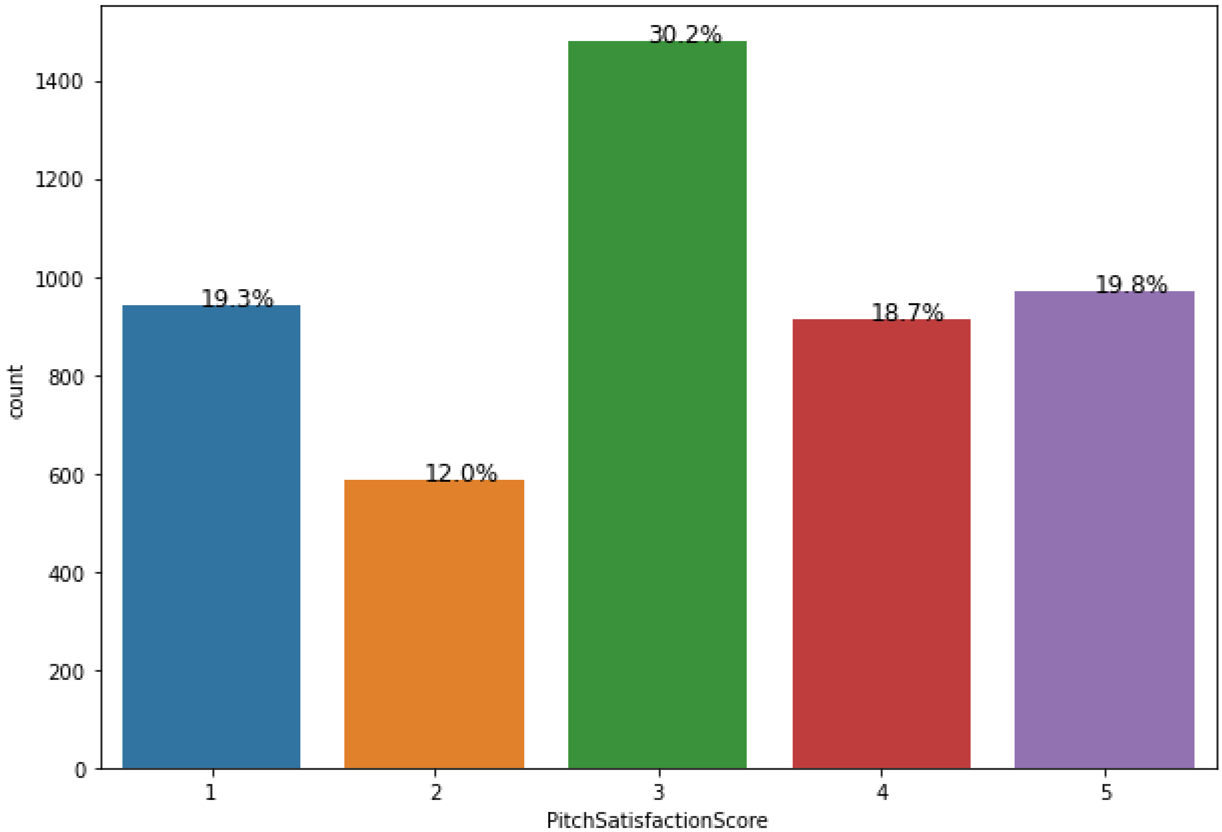


- Most frequent Passport is False(0) with 71%.
- Passports True(1) is 29%.
- There are 2 unique values.

Observations on PitchSatisfactionScore

In [38]:
```
print('PitchSatisfactionScore\n' , data['PitchSatisfactionScore'].value_counts(normaliz
bar_count_pct(data.PitchSatisfactionScore)
```

```
PitchSatisfactionScore
 3    0.302373
5    0.198445
1    0.192717
4    0.186579
2    0.119885
Name: PitchSatisfactionScore, dtype: float64

Top:3
Freq:1478
```

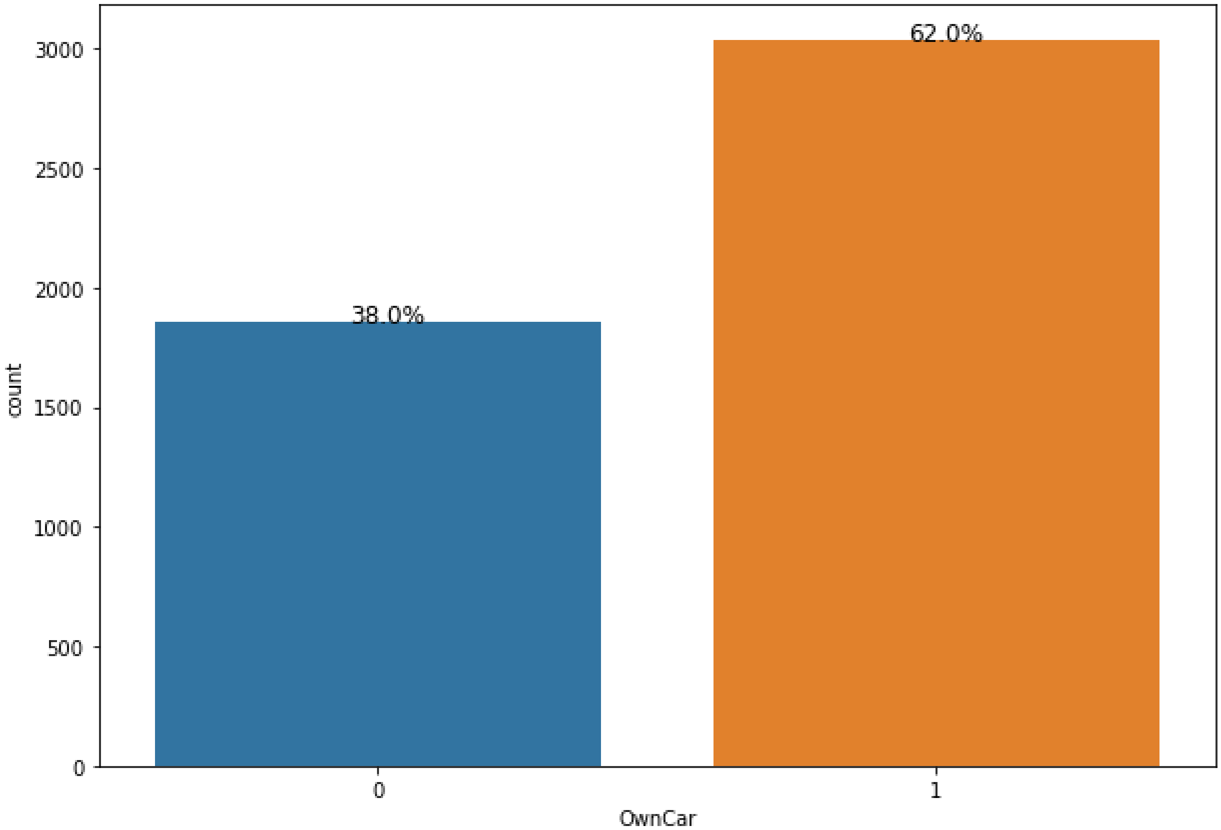


- Most frequent PitchSatisfactionScore is 3 with 30%.
- PitchSatisfactionScore 2 is the lowest with 12%.
- There are 5 unique values.

Observations on OwnCar

In [39]:
```
print('OwnCar\n' , data['OwnCar'].value_counts(normalize=True) , '\n')
bar_count_pct(data.OwnCar)
```

```
OwnCar
 1    0.620295
0    0.379705
Name: OwnCar, dtype: float64

Top:1
Freq:3032
```



- Most frequent OwnCar is True(1) with 62%.
- OwnCar False(0) is 38%.
- There are 2 unique values.

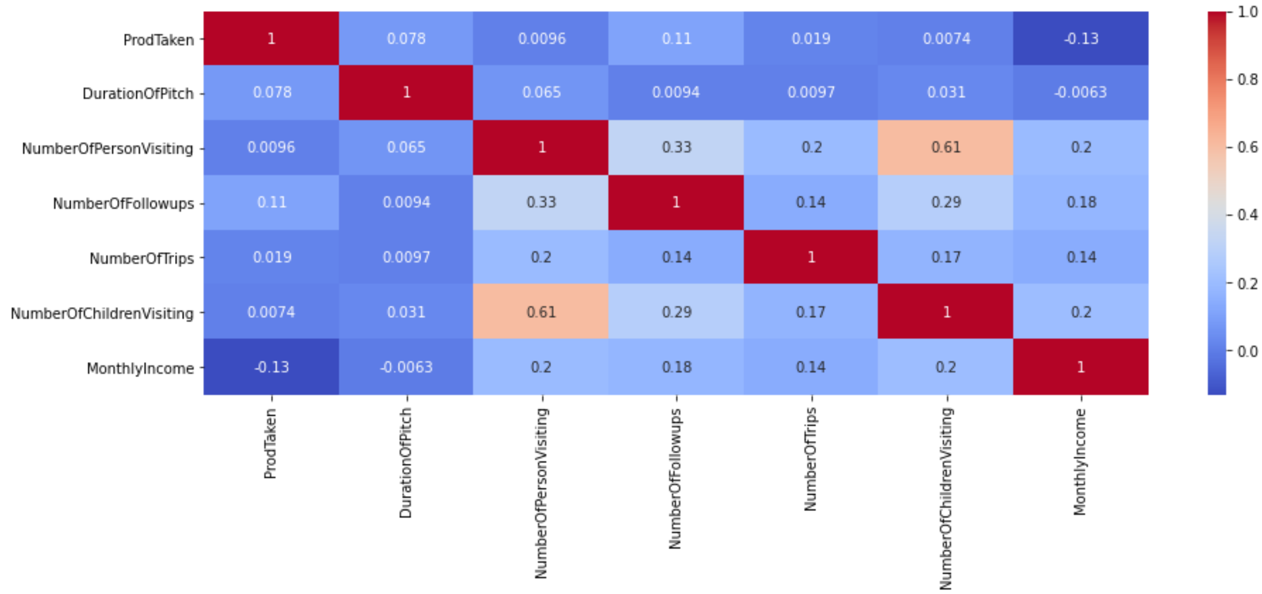
In [ ]:

Bivariate Analysis

In [ ]:

In [40]:

```
plt.figure(figsize=(15,5))
sns.heatmap(data.corr(),annot=True, cmap="coolwarm")
plt.show()
```



- The pairs that have a good/high correlation are:
    - NumberOfChildrenVisiting/NumberOfPersonVisiting
    - NumberOfFollowups/NumberOfPersonVisiting
    - NumberOfFollowups/NumberOfChildrenVisiting
    - NumberOfTrips/NumberOfPersonVisiting
    - NumberOfChildrenVisiting/MonthlyIncome
    - NumberOfPersonVisiting/MonthlyIncome

In [ ]:

```
```

Bivariate analysis every possible attribute pair in relation to ProdTaken

In [41]:

```
### Function to plot distributions and Boxplots of customers
def dist_catplot(x,target='ProdTaken'):
    fig,axs = plt.subplots(2,2,figsize=(12,10))
    axs[0, 0].set_title('Distribution of a customer who take the product')
    sns.distplot(data[(data[target] == 1)][x],ax=axs[0,0],color='teal')
    axs[0, 1].set_title("Distribution of a customer who doesn't take the product")
    sns.distplot(data[(data[target] == 0)][x],ax=axs[0,1],color='orange')
    axs[1,0].set_title('Boxplot with respect to ProdTaken')
    sns.boxplot(data[target],data[x],ax=axs[1,0],palette='gist_rainbow')
    axs[1,1].set_title('Boxplot with respect to ProdTaken - Without outliers')
    sns.boxplot(data[target],data[x],ax=axs[1,1],showfliers=False,palette='gist_rainbow
    plt.tight_layout()
    plt.show()
```

In [42]:

```
### Function to plot stacked bar charts for categorical columns
def stacked_plot(x):
    sns.set()
    ## crosstab
    tab1 = pd.crosstab(x,data['ProdTaken'],margins=True).sort_values(by=1,ascending=Fal
    print(tab1)
    print('-'*120)
    ## visualising the cross tab
    tab = pd.crosstab(x,data['ProdTaken'],normalize='index').sort_values(by=1,ascending
```

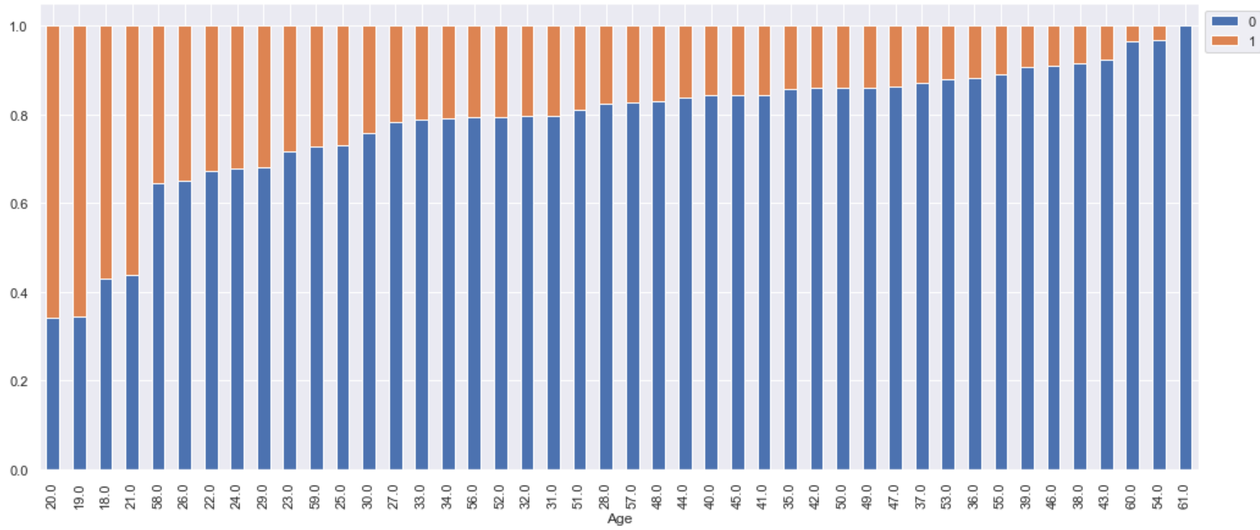
```
    tab.plot(kind='bar',stacked=True,figsize=(17,7))
    plt.legend(loc='lower left', frameon=False,)
    plt.legend(loc="upper left", bbox_to_anchor=(1,1))
    plt.show()
```

Age vs ProdTaken

In [43]:

```
stacked_plot(data['Age'])
```

```
ProdTaken      0    1   All
Age
All         3786  876  4662
29.0         121   57   178
30.0         151   48   199
34.0         167   44   211
31.0         162   41   203
33.0         149   40   189
32.0         157   40   197
26.0          69   37   106
35.0         203   34   237
27.0         108   30   138
36.0         204   27   231
28.0         121   26   147
20.0          13   25    38
41.0         131   24   155
37.0         161   24   185
40.0         123   23   146
21.0          18   23    41
19.0          11   21    32
25.0          54   20    74
42.0         122   20   142
24.0          38   18    56
45.0          98   18   116
44.0          88   17   105
51.0          73   17    90
38.0         161   15   176
22.0          31   15    46
39.0         136   14   150
52.0          54   14    68
23.0          33   13    46
47.0          76   12    88
56.0          46   12    58
50.0          74   12    86
59.0          32   12    44
58.0          20   11    31
48.0          54   11    65
46.0         110   11   121
43.0         120   10   130
49.0          56    9    65
53.0          58    8    66
18.0           6    8    14
55.0          57    7    64
57.0          24    5    29
54.0          59    2    61
60.0          28    1    29
61.0           9    0     9
------------------------------------------------------------------------------------------
------------------------------
```

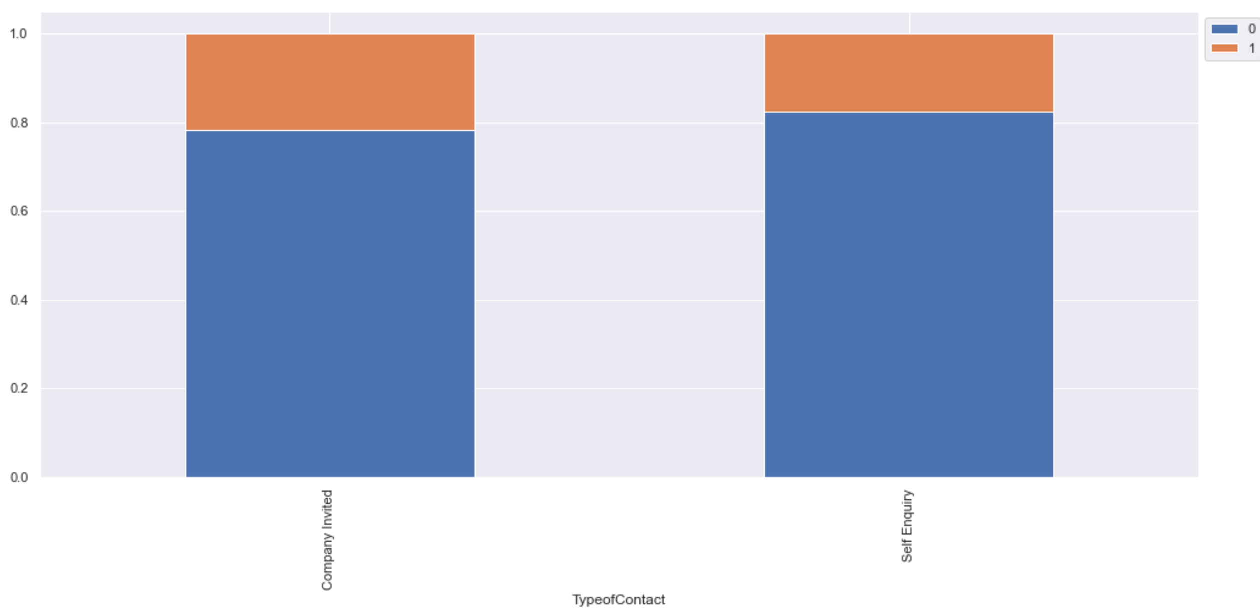


- Propertianlally, There is significat difference in `Ages` 18-22 for customer that take the product is a higher success rate.

TypeofContact vs Prod Taken

In [44]:
```
stacked_plot(data['TypeofContact'])
```

```
ProdTaken          0    1   All
TypeofContact
All             3946  917  4863
Self Enquiry    2837  607  3444
Company Invited 1109  310  1419
------------------------------------------------------------------------------------------------------------------------
```

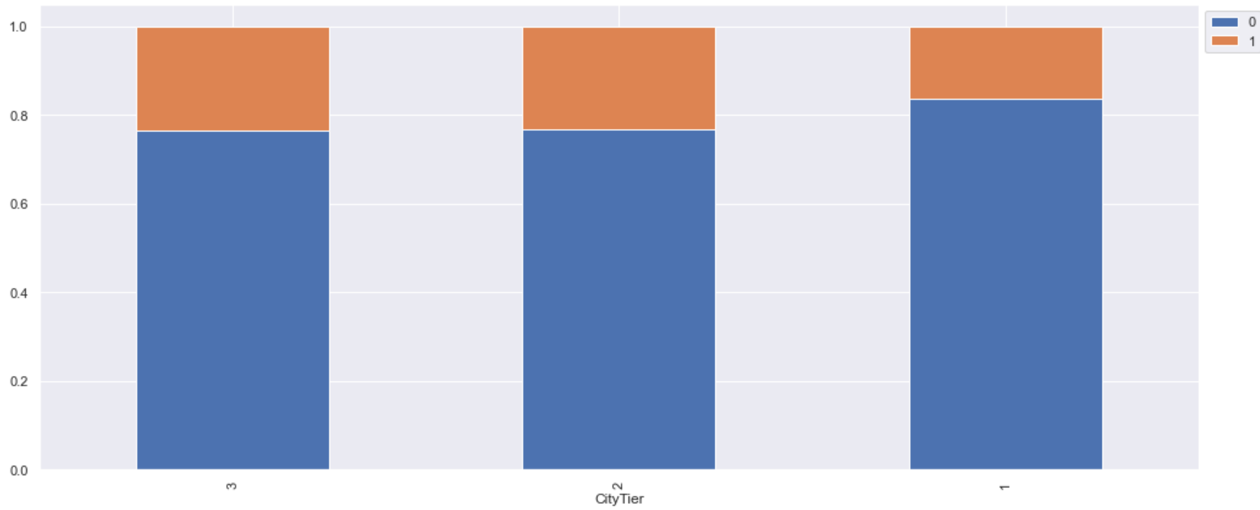


- Proportionaly, there is no significat difference in `TypeofContact` for customer that take the product and those who do not.

CityTier vs Prod Taken

In [45]:
```
stacked_plot(data['CityTier'])
```

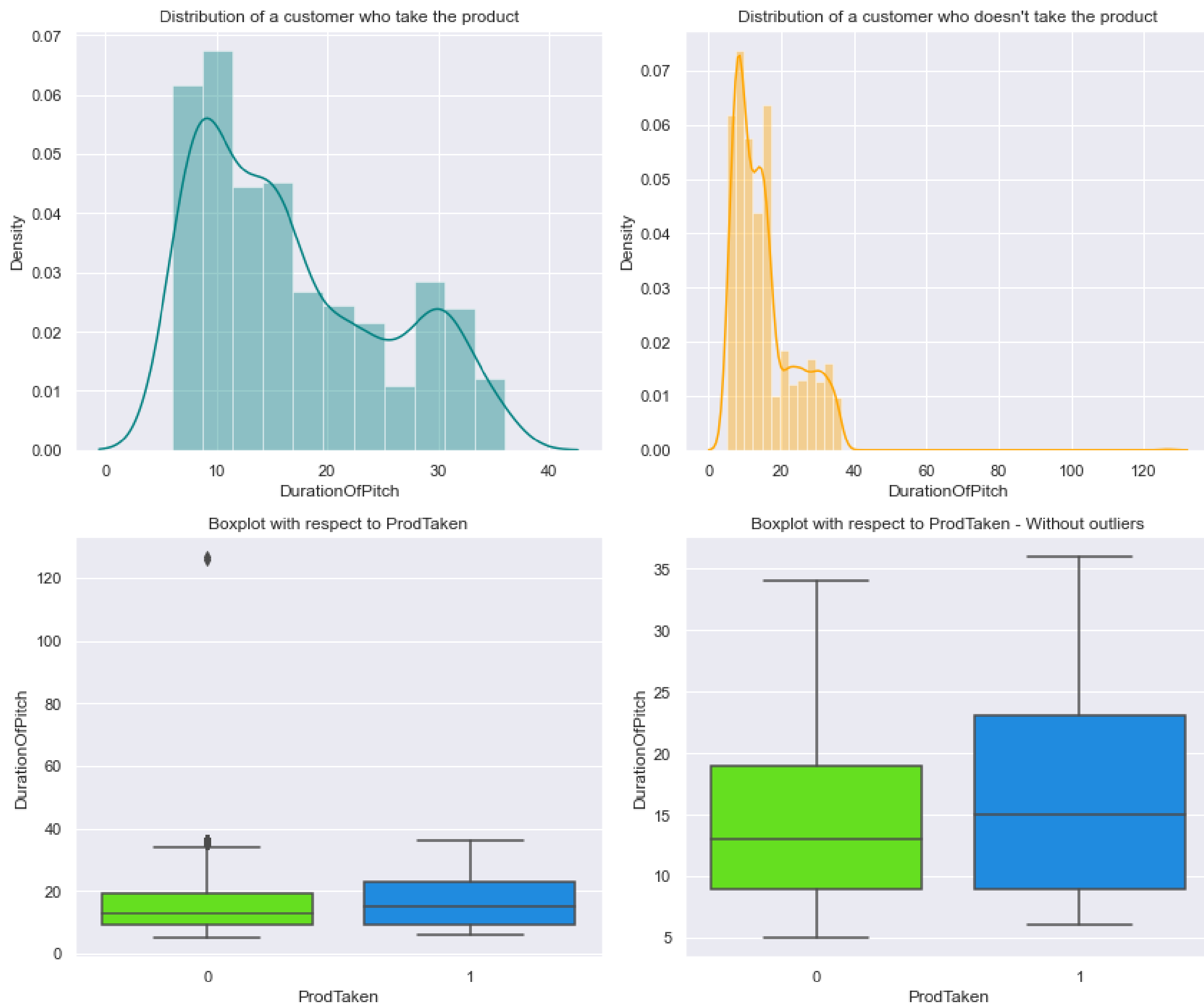
```
ProdTaken     0    1   All
CityTier
All        3968  920  4888
1          2670  520  3190
3          1146  354  1500
2           152   46   198
------------------------------------------------------------------------------------------------------------------------------
```



- Proportionaly, there is no significat difference in `CityTier` for customer that take the product and those who do not.

DurationOfPitch vs Prod Taken

In [46]:
```
dist_catplot('DurationOfPitch')
```

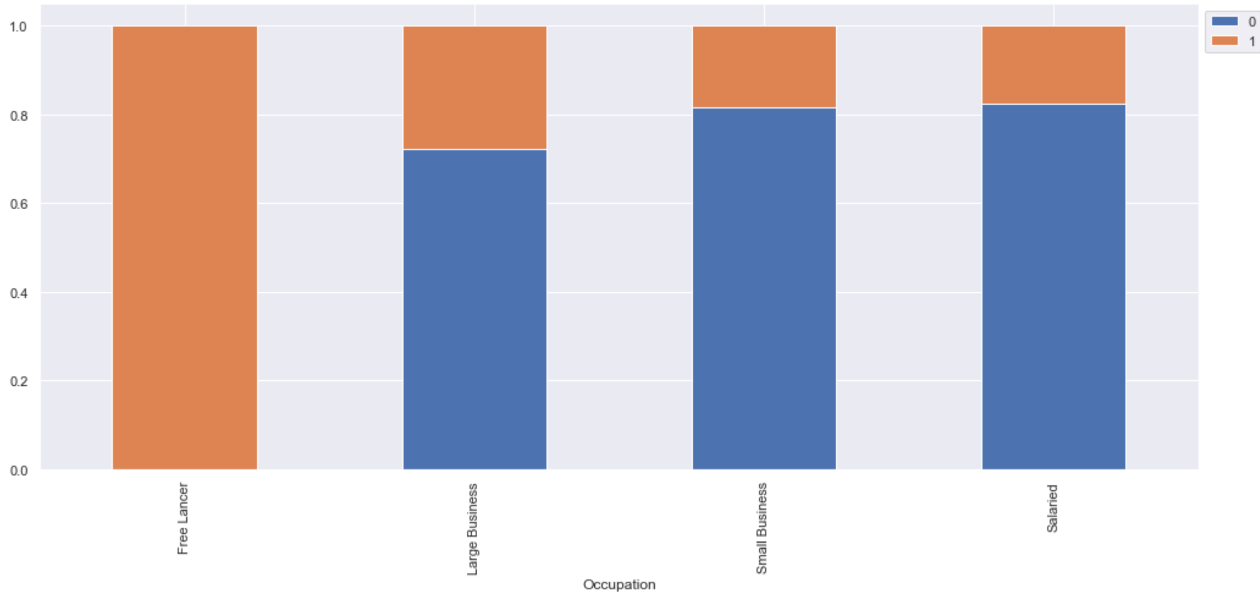


- When outliers are included in the comparison, there does not seem to be major difference in `Duration of Pitch` in regards to ProdTaken.
- When we remove outliers from the plot, we can see slight increase in ProdTaken when `DurationOfPitch` is higher than 20.

Occupation vs Prod Taken

In [47]:
```
stacked_plot(data['Occupation'])
```

```
ProdTaken          0    1   All
Occupation
All             3968  920  4888
Salaried        1954  414  2368
Small Business  1700  384  2084
Large Business   314  120   434
Free Lancer        0    2     2
------------------------------------------------------------------------------------
------------------------------
```



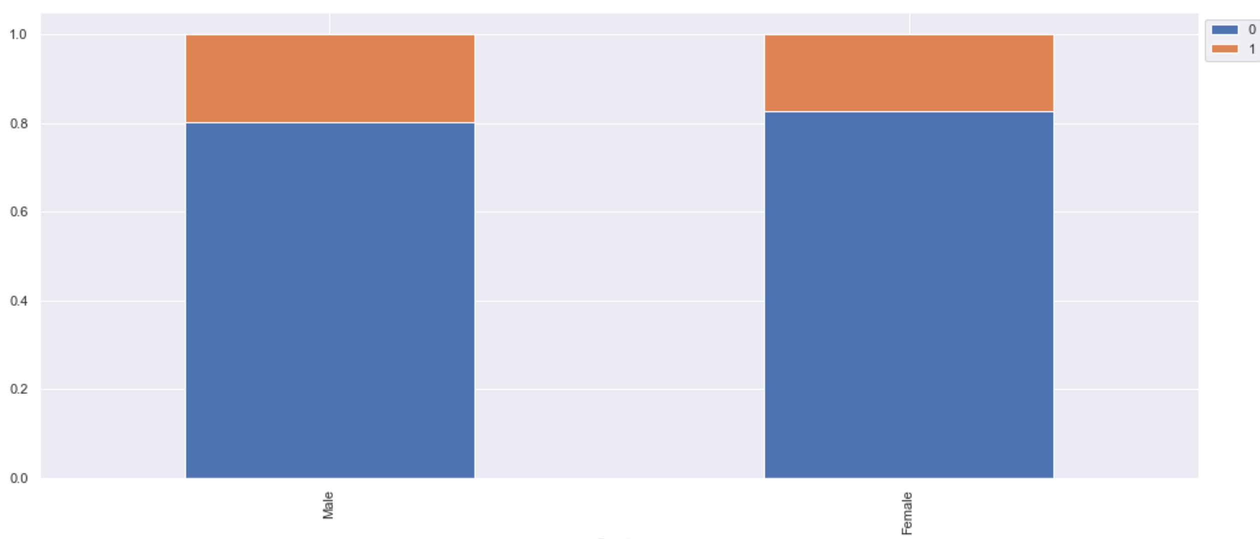
- Proportionaly, there is no significat difference in `Occupation` bewteen the 'Large Business' , 'Small Business' and ' Salaried' categories.
- There is however a significant different in the 'Free Lancer' category wtih 0%. But this category only has 2 customers which is not impactfull in the overall picture.

Gender vs Prod Taken

In [48]:

```
stacked_plot(data['Gender'])
```

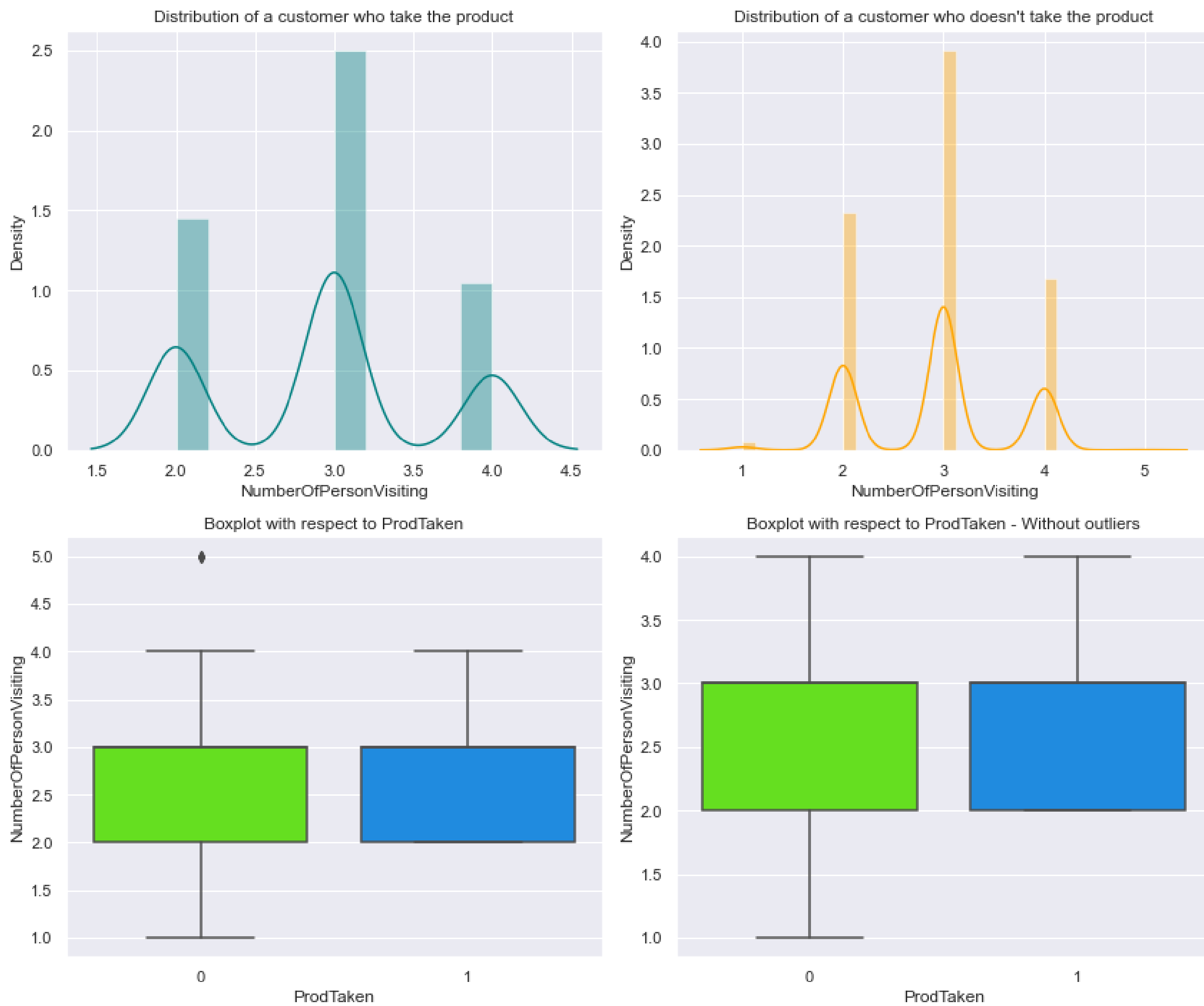
```
ProdTaken     0    1   All
Gender
All        3968  920  4888
Male       2338  578  2916
Female     1630  342  1972
------------------------------------------------------------------------------------------
--------------------------------
```



- Proportionaly, there is no significat difference in `Gender` for customer that take the product and those who do not.

NumberOfPersonVisiting vs Prod Taken

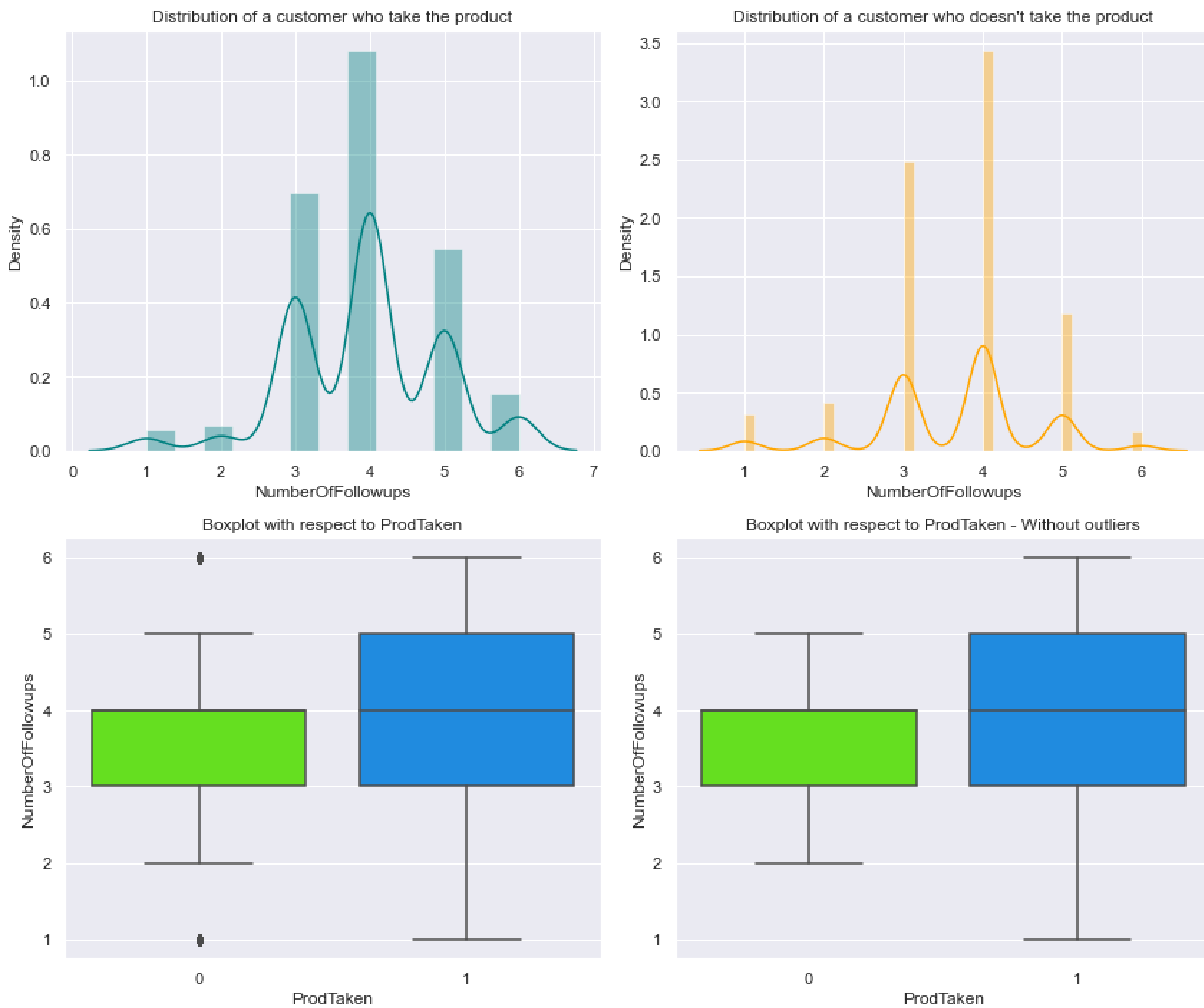
In [49]:
```
dist_catplot('NumberOfPersonVisiting')
```



- There is no significant difference on `NumerOfPersonVisiting` when compared with or without outliers. In regards to ProdTaken.

NumberOfFollowups vs Prod Taken

In [50]:
```
dist_catplot('NumberOfFollowups')
```

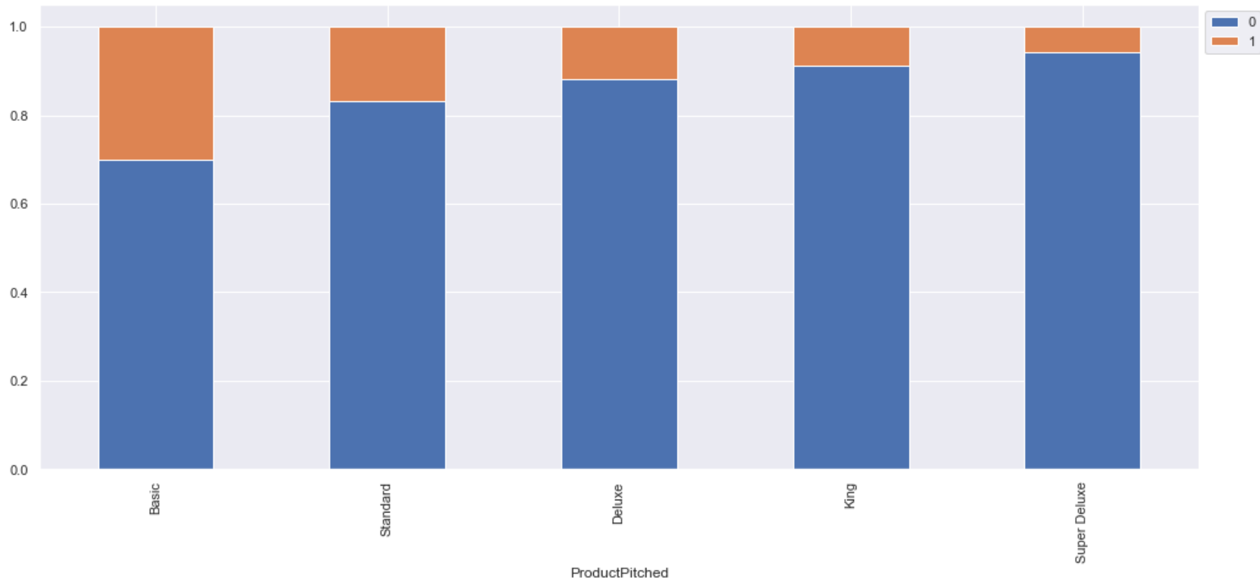


- There is no significant difference on `NumberOfFollowups` when compared with or without outliers. In regards to ProdTaken.
- There is, however, an increase in ProdTaken in the ranges of 4-5 when compared to customer that did not ProdTaken.

ProductPitched vs Prod Taken

In [51]:
```
stacked_plot(data['ProductPitched'])
```

```
ProdTaken          0    1   All
ProductPitched
All             3968  920  4888
Basic           1290  552  1842
Deluxe          1528  204  1732
Standard         618  124   742
King             210   20   230
Super Deluxe     322   20   342
------------------------------------------------------------------------------------------
--------------------------------
```

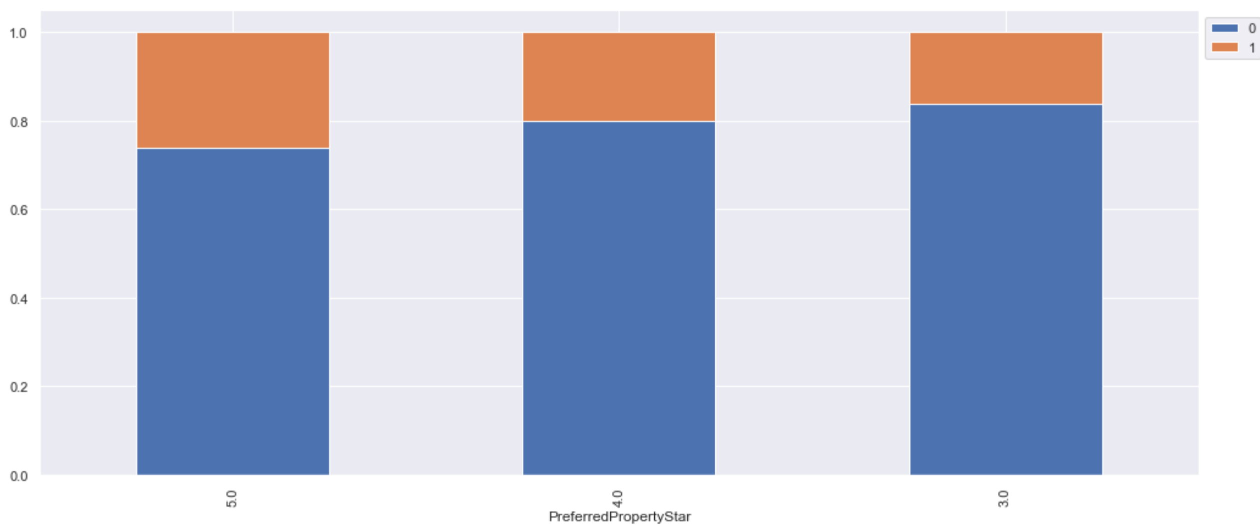


- Proportionaly, there is a slight increase in `ProductPitched` Basic. This product is also the most pitched.

PreferredPropertyStar vs Prod Taken

In [52]:
```
stacked_plot(data['PreferredPropertyStar'])
```

```
ProdTaken                 0    1   All
PreferredPropertyStar
All                    3948  914  4862
3.0                    2511  482  2993
5.0                     706  250   956
4.0                     731  182   913
------------------------------------------------------------------------------------------------------------------------
```

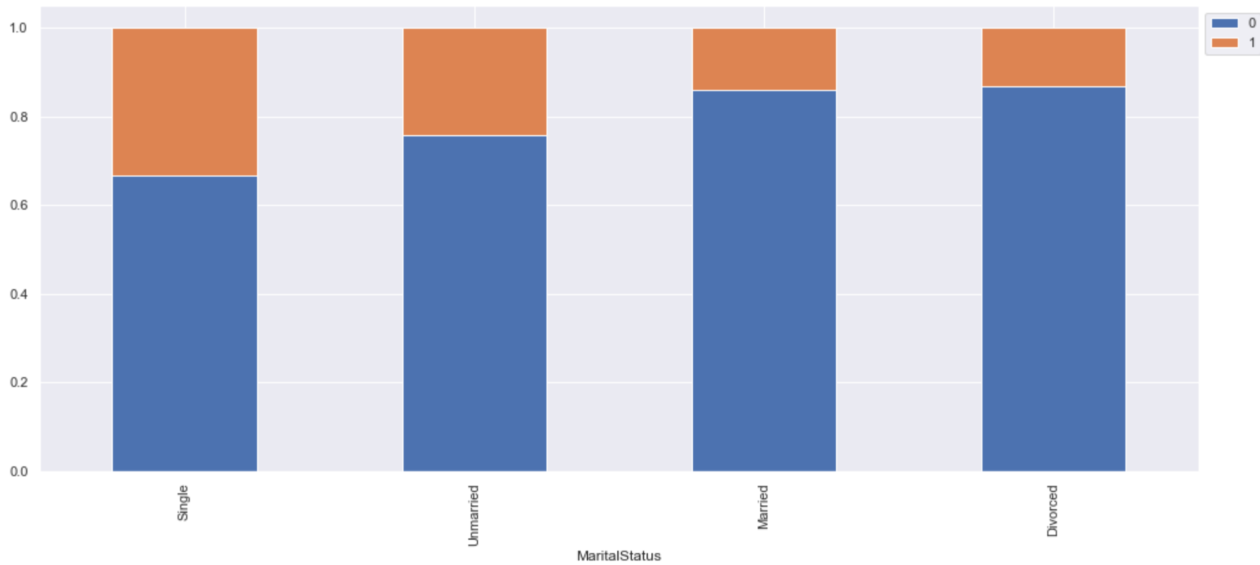


- Proportionaly, there is no differenct between all the PreferredPropertyStar categories.
- PreferredPropertyStar is the most common selected.

MaritalStatus vs Prod Taken

In [53]:
```
stacked_plot(data['MaritalStatus'])
```

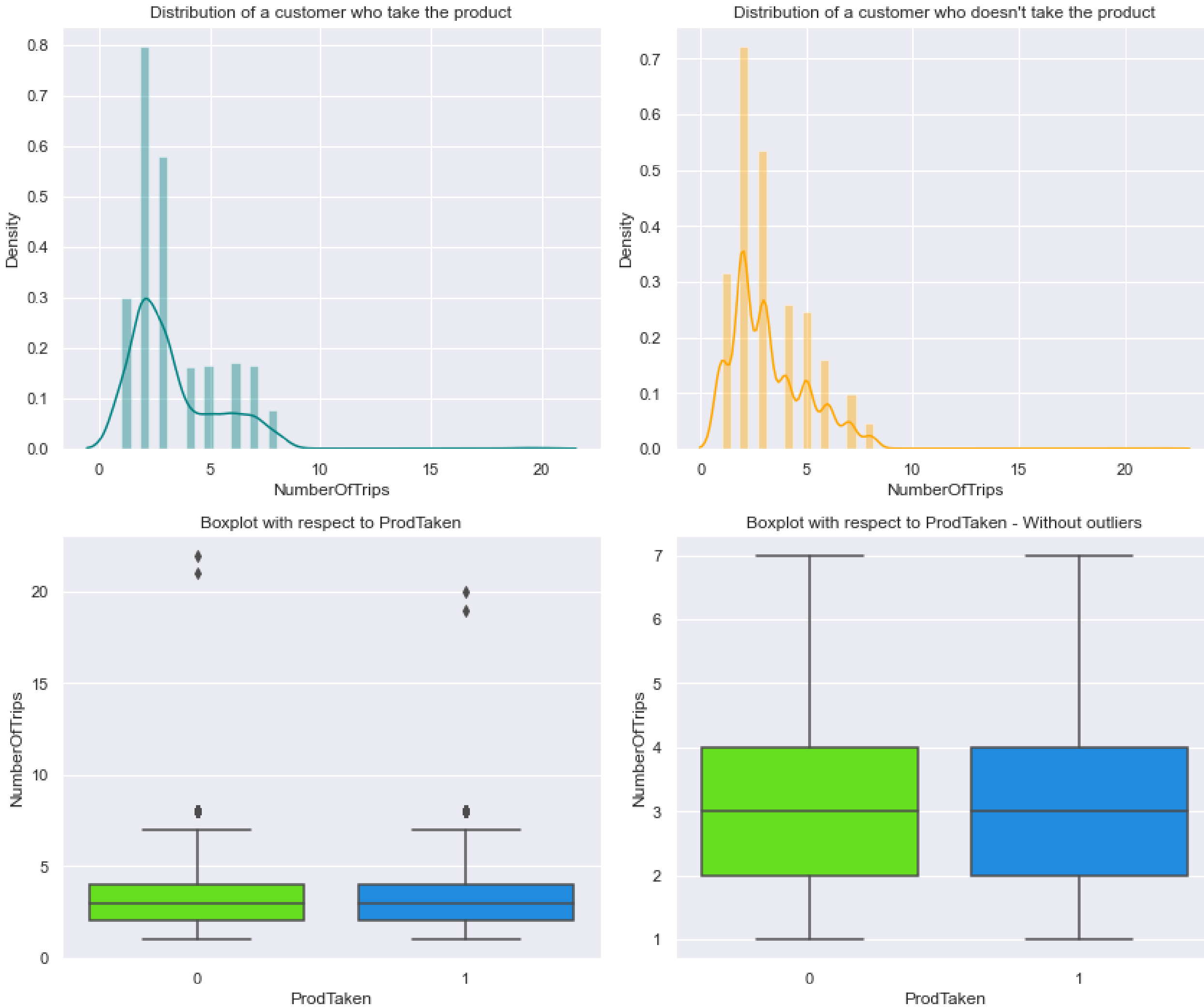
```
ProdTaken         0    1   All
MaritalStatus
All            3968  920  4888
Married        2014  326  2340
Single          612  304   916
Unmarried       516  166   682
Divorced        826  124   950
------------------------------------------------------------------------------------------------------------------------------
```



- Proportionaly, Single category in `MaritalStatus` is the most successfull.

NumberOfTrips vs Prod Taken

In [54]:
```
dist_catplot('NumberOfTrips')
```



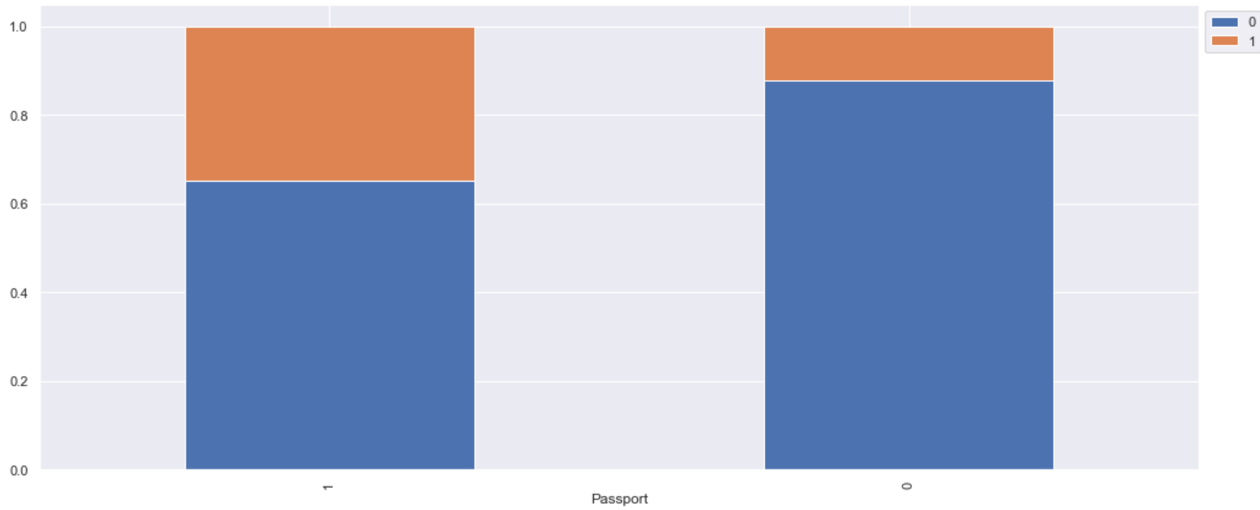
- There is no major difference in `NumberOfTrips` in regards to ProdTaken.

Passport vs Prod Taken

In [55]:

```
stacked_plot(data['Passport'])
```

```
ProdTaken     0    1   All
Passport
All        3968  920  4888
1           928  494  1422
0          3040  426  3466
------------------------------------------------------------------------------------------
--------------------------------
```



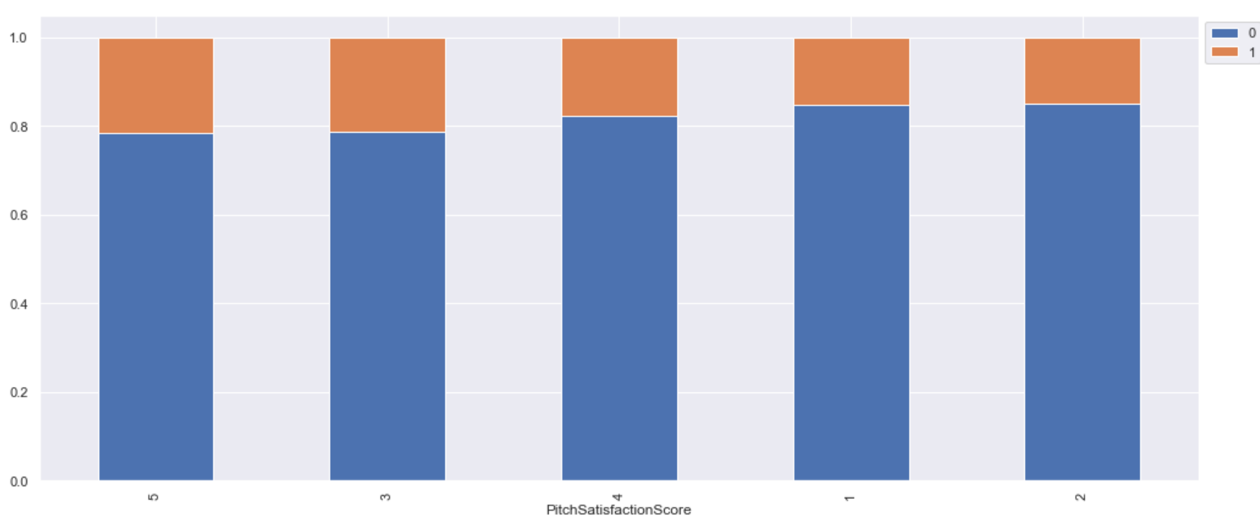
- Customers that have `Passport` have a higher success rate on ProdTaken.

PitchSatisfactionScore vs Prod Taken

In [56]:

```
stacked_plot(data['PitchSatisfactionScore'])
```

```
ProdTaken                  0    1   All
PitchSatisfactionScore
All                     3968  920  4888
3                       1162  316  1478
5                        760  210   970
4                        750  162   912
1                        798  144   942
2                        498   88   586
------------------------------------------------------------------------------------------
--------------------------------
```



- There is no major diference in `PitchSatisfactionScore` with regard to ProdTaken.

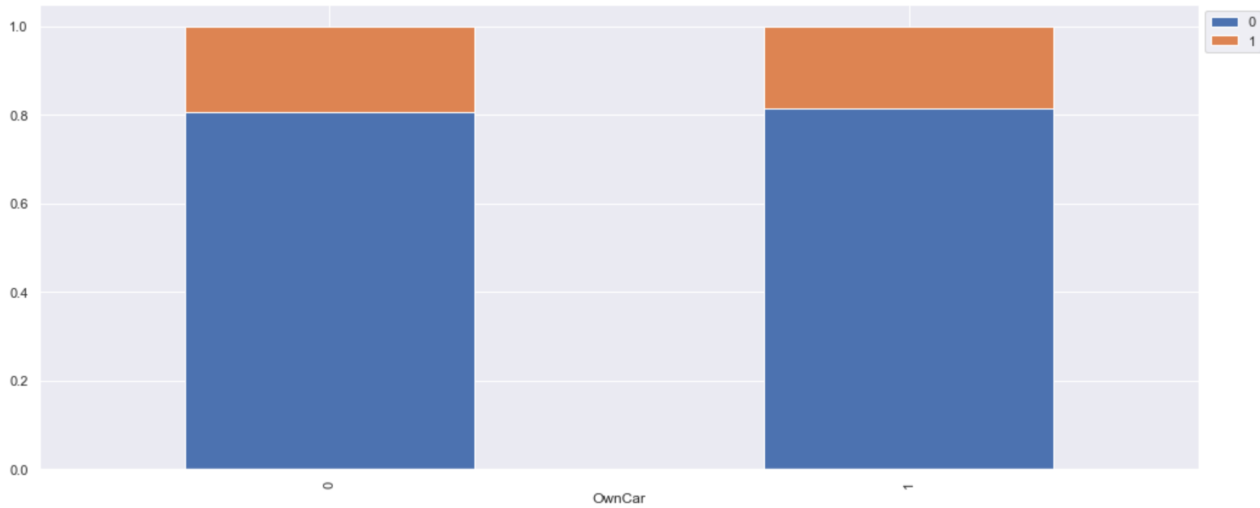
OwnCar vs Prod Taken

In [57]:

```
stacked_plot(data['OwnCar'])
```

```
ProdTaken     0    1   All
OwnCar
All        3968  920  4888
```

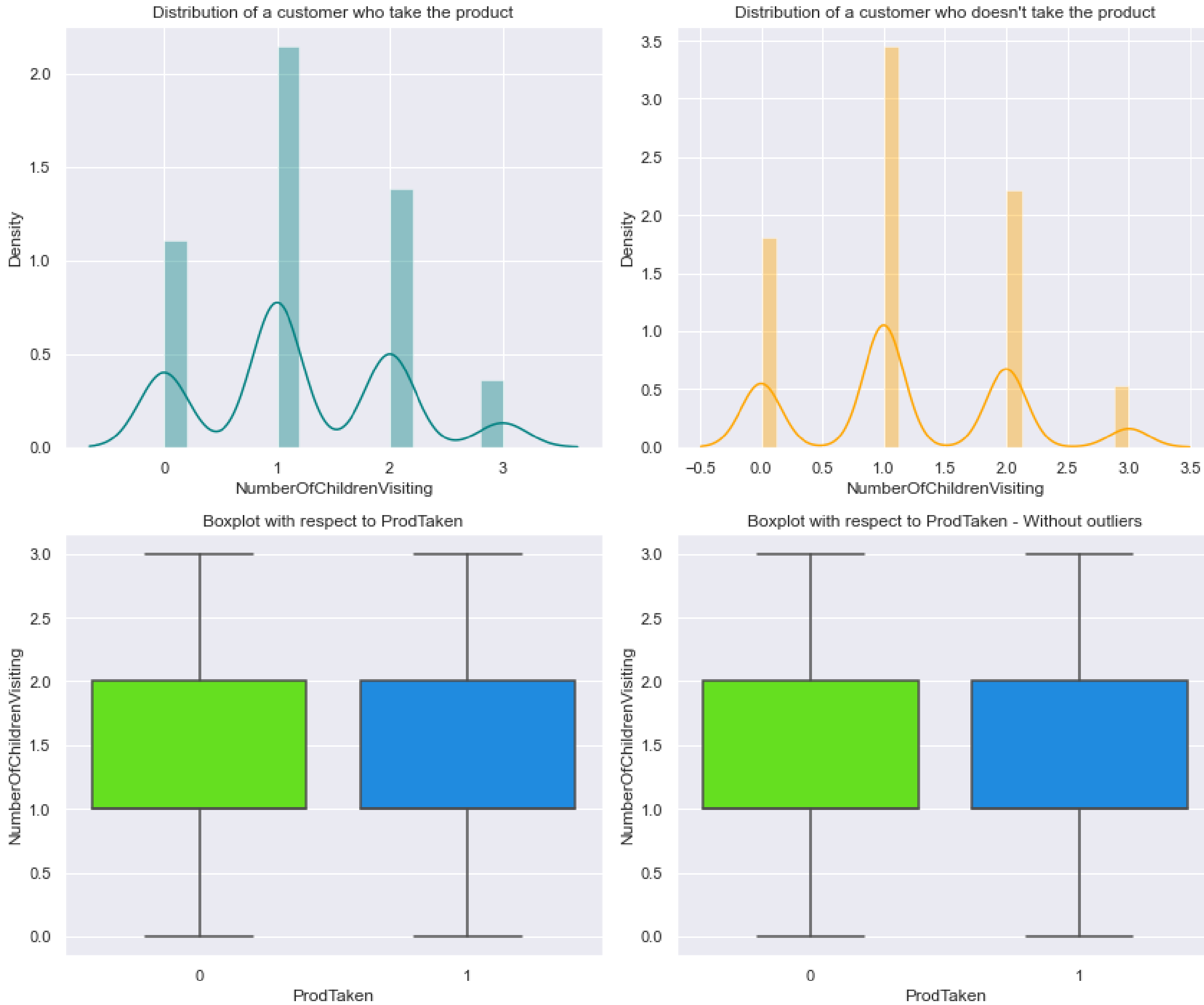
```
1           2472  560  3032
0           1496  360  1856
------------------------------------------------------------------------------------------
--------------------------------
```



- There is no major diference in `OwnCar` with regard to ProdTaken.

NumberOfChildrenVisiting vs Prod Taken

In [58]:
```
dist_catplot('NumberOfChildrenVisiting')
```

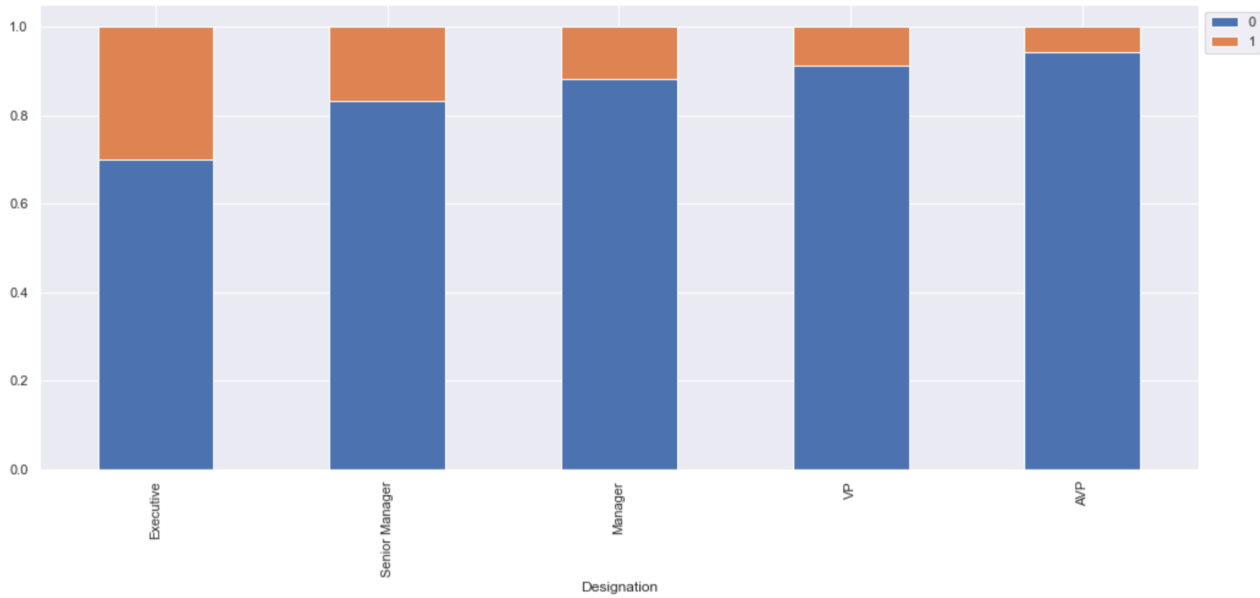


- There is no major diference in `NumberOfChildrenVisiting` with regard to ProdTaken.

Designation vs Prod Taken

In [59]:
```
stacked_plot(data['Designation'])
```

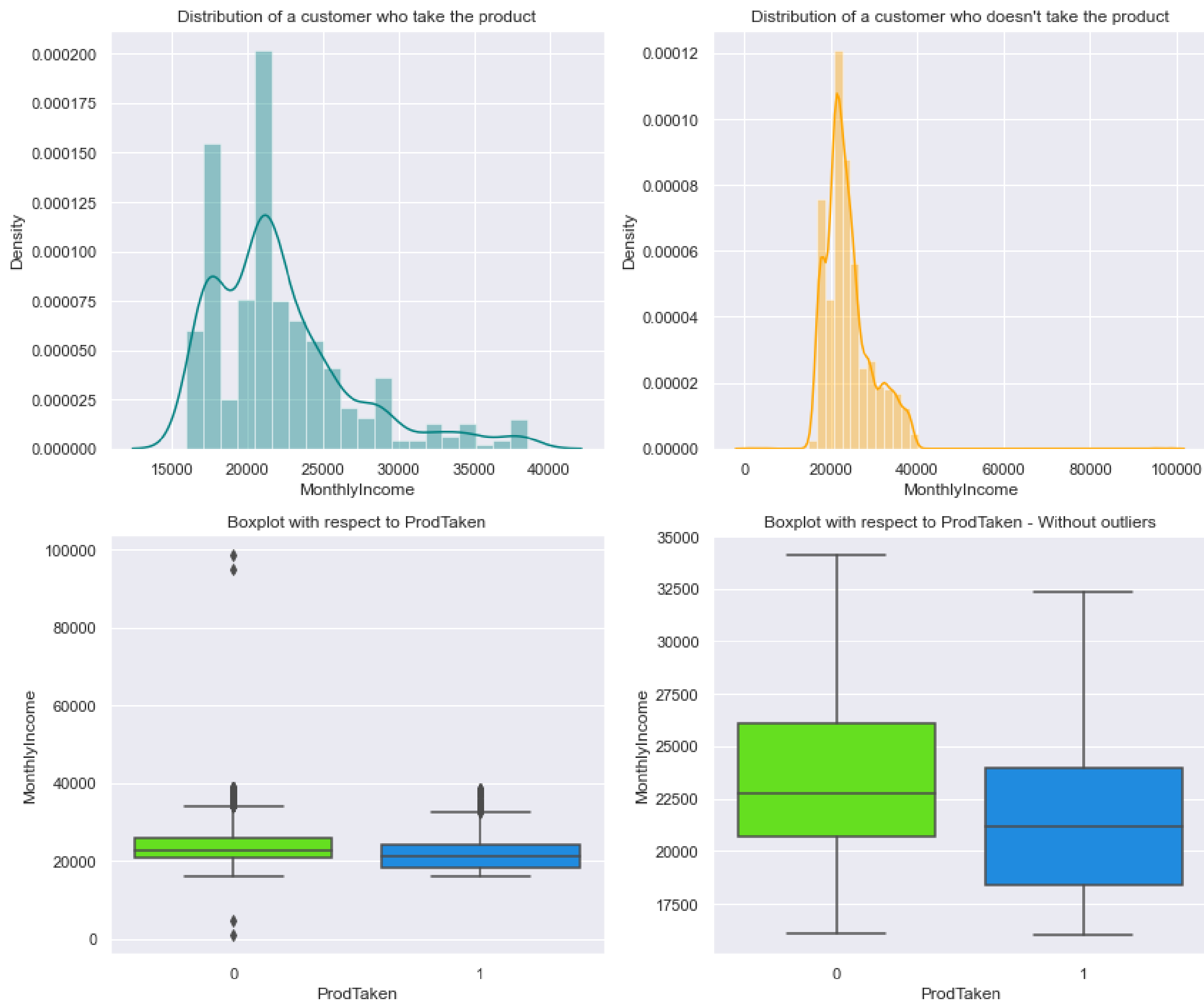
```
ProdTaken          0    1   All
Designation
All             3968  920  4888
Executive       1290  552  1842
Manager         1528  204  1732
Senior Manager   618  124   742
AVP              322   20   342
VP               210   20   230
------------------------------------------------------------------------------------------
--------------------------------
```



- Executive category for `Designation` is the most successful in regards to ProdTaken.
- All other categories seem to perform equally.

MonthlyIncome vs Prod Taken

In [60]:
```
dist_catplot('MonthlyIncome')
```



- Those customers who have an income higher than 20k-27k dollars are customers who will not take ProdTaken.
- Income seems to be a significant predictor as it provides good separation between two classes.

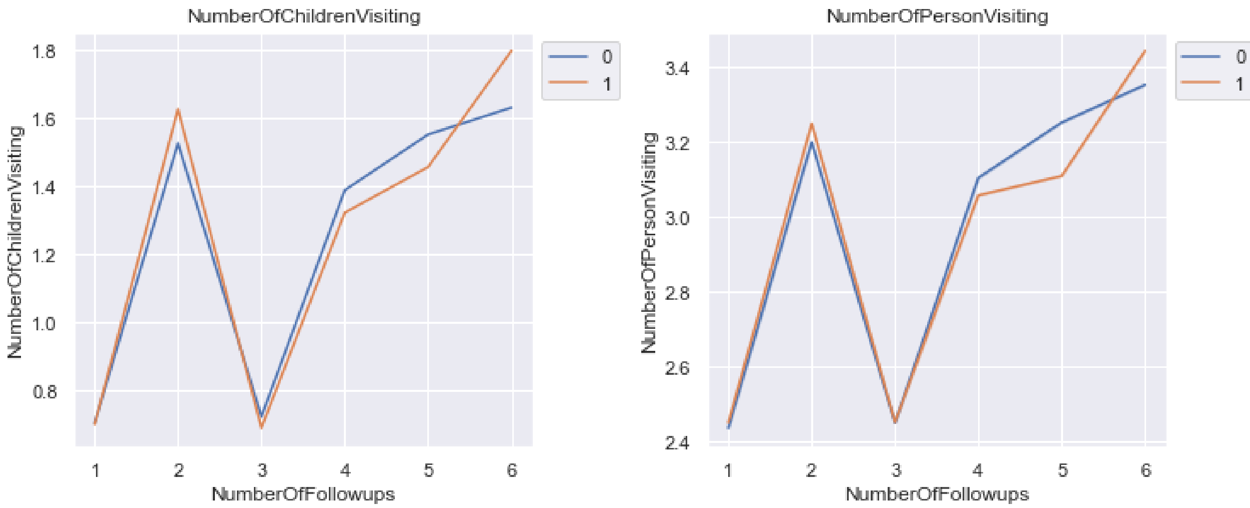
Multivariate analysis

Analys variables with good and high correlation with regards to ProdTaken.

- NumberOfChildrenVisiting/NumberOfPersonVisiting
- NumberOfFollowups/NumberOfPersonVisiting
- NumberOfFollowups/NumberOfChildrenVisiting
- NumberOfTrips/NumberOfPersonVisiting
- NumberOfChildrenVisiting/MonthlyIncome
- NumberOfPersonVisiting/MonthlyIncome

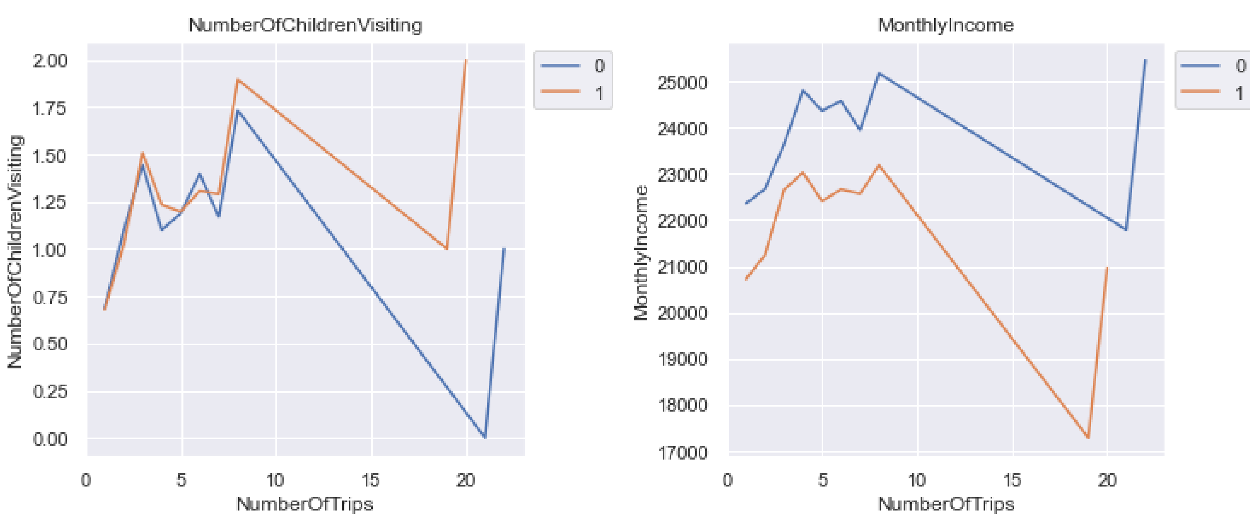
In [61]:
```
cols = data[['NumberOfChildrenVisiting','NumberOfPersonVisiting']].columns.tolist()
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                     plt.subplot(3,3,i+1)
                     sns.lineplot(data['NumberOfFollowups'],data[variable],hue=data['Pr
```

```
                     plt.tight_layout()
                     plt.title(variable)
                     plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



- When visualizing difference bewteen NumberOfChildrenVisiting and NumberOfPersonVisiting in regards to NumberOfFollowups and Prod Taken, there really is no significant difference. This could be a sign of Multicollinearity.

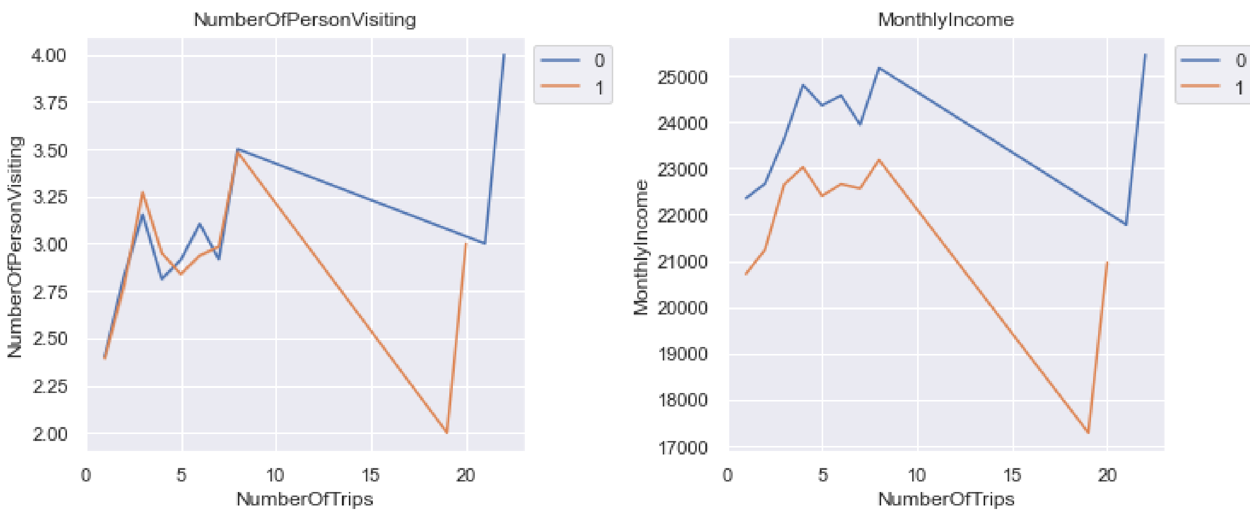
In [62]:
```
cols = data[['NumberOfChildrenVisiting','MonthlyIncome']].columns.tolist()
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                     plt.subplot(3,3,i+1)
                     sns.lineplot(data['NumberOfTrips'],data[variable],hue=data['ProdTa
                     plt.tight_layout()
                     plt.title(variable)
                     plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



When visualizing difference bewteen NumberOfChildrenVisiting and NumberOfPersonVisiting in regards to NumberOfTrips and Prod Taken, there seems to be a major difference on the extreme(outlier) cases. This may not be influential, otherwise this may ba a case of multicollinearity.

In [63]:
```
cols = data[['NumberOfPersonVisiting','MonthlyIncome']].columns.tolist()
```

```
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                     plt.subplot(3,3,i+1)
                     sns.lineplot(data['NumberOfTrips'],data[variable],hue=data['ProdTa
                     plt.tight_layout()
                     plt.title(variable)
                     plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



- There is a difference between NumberOfPersonsVisiting and MonthlyIncome, in regards to NumberOfTrips and ProdTaken.

In [ ]:

Data Pre-processing

Null treatment

- Out of 4888 rows, the following columns have null values

In [64]:
```
data.isnull().sum()
```

Out[64]:
```
ProdTaken                    0
Age                        226
TypeofContact               25
CityTier                     0
DurationOfPitch            251
Occupation                   0
Gender                       0
NumberOfPersonVisiting       0
NumberOfFollowups           45
ProductPitched               0
PreferredPropertyStar       26
MaritalStatus                0
NumberOfTrips              140
Passport                     0
PitchSatisfactionScore       0
OwnCar                       0
NumberOfChildrenVisiting    66
Designation                  0
```

```
MonthlyIncome              233
dtype: int64
```

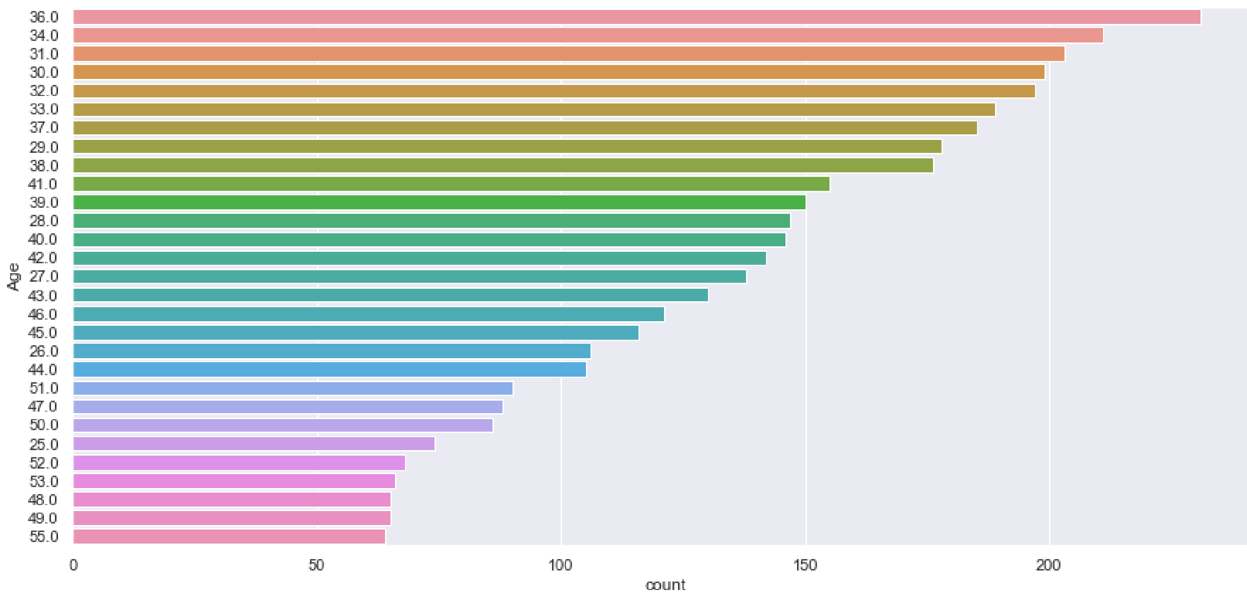
Treat Age nulls

In [65]:
```python
data.Age.unique()
```

Out[65]:
```
[41.0, 49.0, 37.0, 33.0, NaN, ..., 52.0, 47.0, 18.0, 60.0, 61.0]
Length: 45
Categories (44, float64): [41.0, 49.0, 37.0, 33.0, ..., 47.0, 18.0, 60.0, 61.0]
```

In [66]:
```python
plt.figure(figsize=(15, 7))
sns.countplot(y="Age", data=data, order=data["Age"].value_counts().index[1:30])
```

Out[66]:
```
<AxesSubplot:xlabel='count', ylabel='Age'>
```



- There are 45 difference ages, we will try to reduce this and group by AgeRanges.
- First, we must treat null values and outliers
- We will treat null values by replacing these with the Mode.

In [67]:
```python
data.Age.mode()
```

Out[67]:
```
0    35.0
Name: Age, dtype: category
Categories (44, float64): [18.0, 19.0, 20.0, 21.0, ..., 58.0, 59.0, 60.0, 61.0]
```

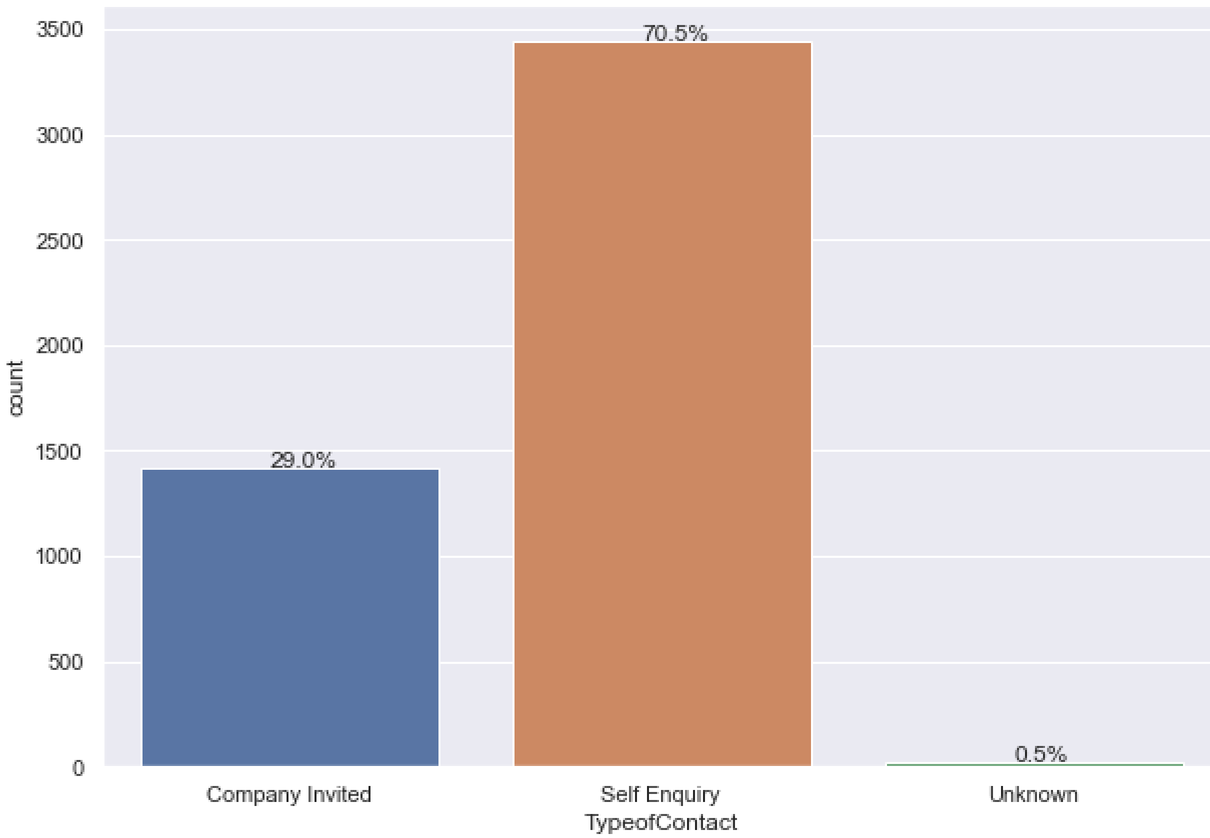
In [68]:
```python
data['Age'] = data['Age'].fillna(data['Age'].mode()[0])
```

Treat TypeofContact nulls

In [69]:
```python
data['TypeofContact'] = data['TypeofContact'].cat.add_categories('Unknown')
data['TypeofContact'].fillna('Unknown', inplace =True)
```

In [70]:
```python
bar_count_pct(data.TypeofContact)
```

```
Top:Self Enquiry
Freq:3444
```



Treat DurationOfPitch nulls

In [71]:
```
data['DurationOfPitch'] = data['DurationOfPitch'].fillna(data['DurationOfPitch'].mean()
```

Treat NumberOfFollowups nulls

In [72]:
```
data['NumberOfFollowups'] = data['NumberOfFollowups'].fillna(data['NumberOfFollowups'].
```

Treat PreferredPropertyStar nulls

In [73]:
```
data['PreferredPropertyStar'] = data['PreferredPropertyStar'].fillna(data['PreferredPro
```

Treat NumberOfTrips nulls

In [74]:
```
data['NumberOfTrips'] = data['NumberOfTrips'].fillna(data['NumberOfTrips'].mode()[0])
```

Treat NumberOfChildrenVisiting nulls

In [75]:
```
data['NumberOfChildrenVisiting'] = data['NumberOfChildrenVisiting'].fillna(data['Number
```

Treat MonthlyIncome nulls

In [76]:
```
data['MonthlyIncome'] = data['MonthlyIncome'].fillna(data['MonthlyIncome'].mean())
```

In [ ]:
```

```

In [77]:
```
data.isnull().sum()
```

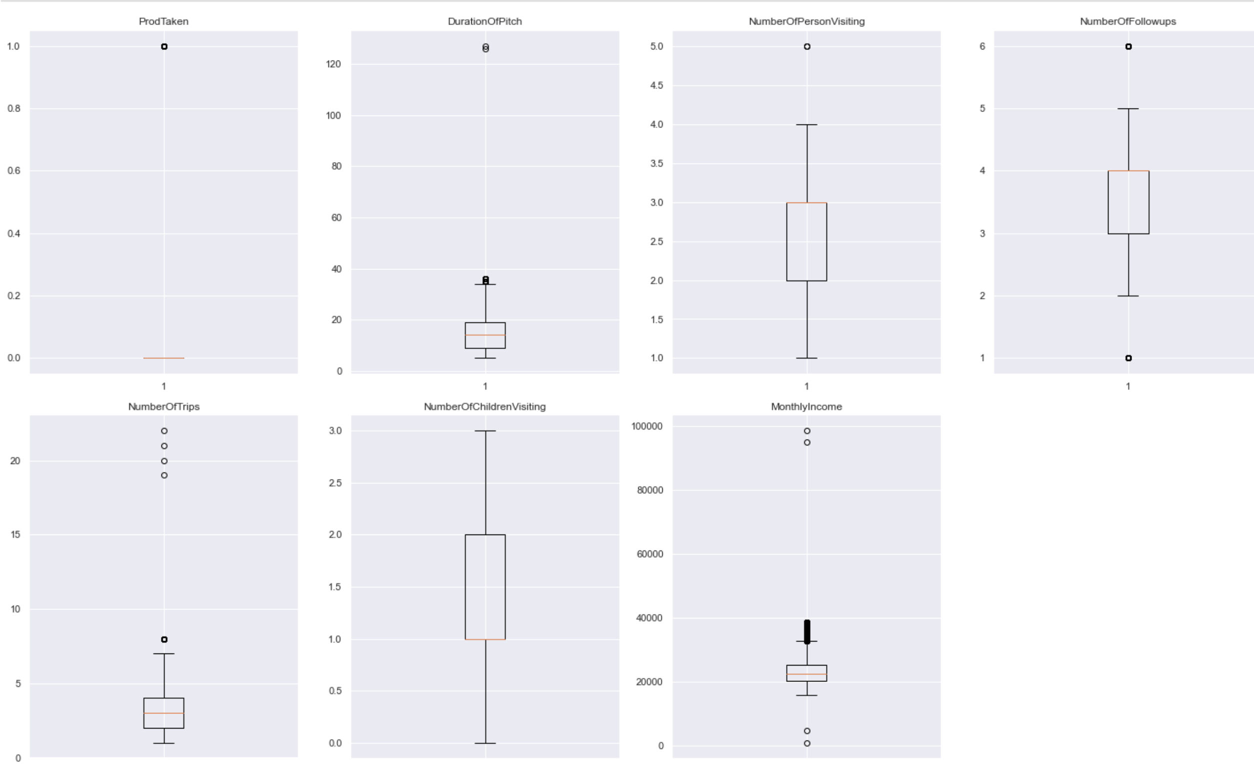
Out[77]:
```
ProdTaken                   0
```

```
Age                        0
TypeofContact              0
CityTier                   0
DurationOfPitch            0
Occupation                 0
Gender                     0
NumberOfPersonVisiting     0
NumberOfFollowups          0
ProductPitched             0
PreferredPropertyStar      0
MaritalStatus              0
NumberOfTrips              0
Passport                   0
PitchSatisfactionScore     0
OwnCar                     0
NumberOfChildrenVisiting   0
Designation                0
MonthlyIncome              0
dtype: int64
```

- There are no null values now

Outlier treatment

In [78]:
```python
numerical_col = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(20,30))

for i, variable in enumerate(numerical_col):
                     plt.subplot(5,4,i+1)
                     plt.boxplot(data[variable],whis=1.5)
                     plt.tight_layout()
                     plt.title(variable)

plt.show()
```

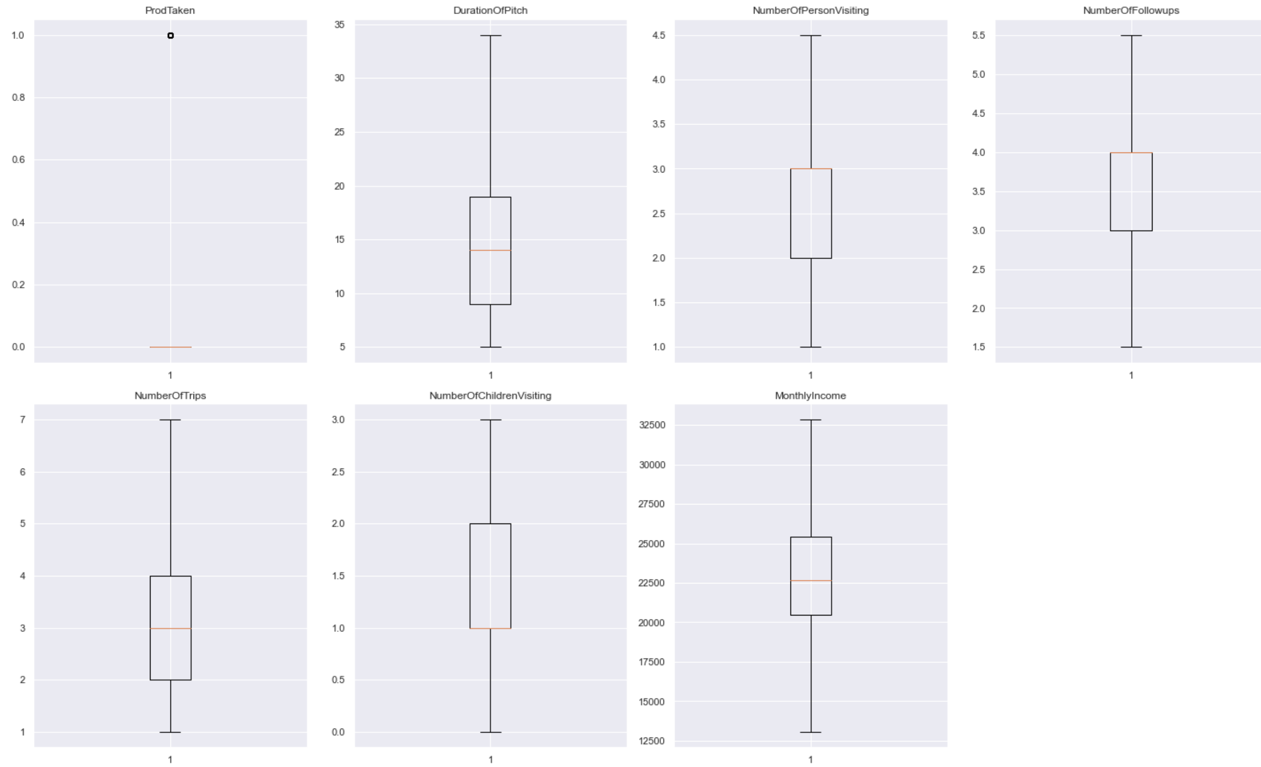


- We will remove `ProdTaken` from outlier treatment since this is a boolean, and also is the dependant variable.

In [79]:
```
def treat_outliers(df,col):
    '''
    treats outliers in a varaible
    col: str, name of the numerical varaible
    df: data frame
    col: name of the column
    '''
    Q1=df[col].quantile(0.25) # 25th quantile
    Q3=df[col].quantile(0.75)  # 75th quantile
    IQR=Q3-Q1
    Lower_Whisker = Q1 - 1.5*IQR
    Upper_Whisker = Q3 + 1.5*IQR
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker) # all the values samller t
                                                            # and all the values above
    return df

def treat_outliers_all(df, col_list):
    '''
    treat outlier in all numerical varaibles
    col_list: list of numerical varaibles
    df: data frame
    '''
    for c in col_list:
        df = treat_outliers(df,c)

    return df
```

In [80]:
```
numerical_col = data.select_dtypes(include=np.number).columns.tolist()# getting list of


# items to be removed
unwanted= {'ProdTaken'} # these column have very few non zero observation , doing outli

numerical_col = [ele for ele in numerical_col if ele not in unwanted]
data = treat_outliers_all(data,numerical_col)
```

In [81]:
```
numerical_col = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(20,30))

for i, variable in enumerate(numerical_col):
                     plt.subplot(5,4,i+1)
                     plt.boxplot(data[variable],whis=1.5)
                     plt.tight_layout()
                     plt.title(variable)

plt.show()
```



- There are no outliers now

Transform Age into bigger groups

In [82]:

```
def age_grouping(age):
        return (age//10)*10
```

In [83]:

```
data['AgeGroup'] = data['Age'].apply(age_grouping)

data.AgeGroup.unique()
```

Out[83]:

```
array([40., 30., 50., 20., 10., 60.])
```

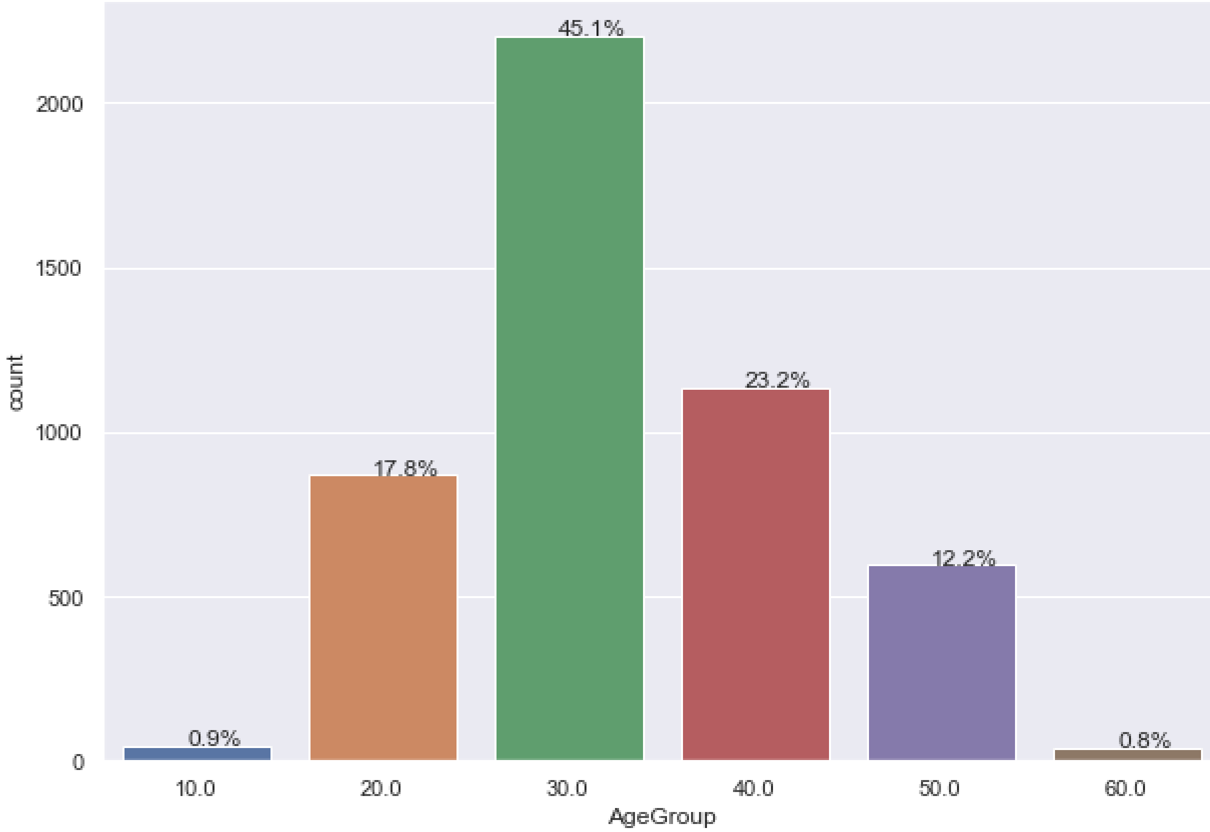
In [84]:

```
data = data.drop(['Age'], axis=1)
```

In [85]:

```
print('AgeGroup\n' , data['AgeGroup'].value_counts(normalize=True) , '\n')
bar_count_pct(data.AgeGroup)
```

```
AgeGroup
 30.0    0.450900
40.0    0.231792
20.0    0.177987
50.0    0.122136
10.0    0.009411
60.0    0.007774
Name: AgeGroup, dtype: float64

Top:30.0
Freq:2204
```



- Now it is more clear as to what age group is most prevalent in the dataset.

- Finally, let convert ProdTaken to category so it is handled correctly

In [86]:
```
data['ProdTaken'] = data.ProdTaken.astype('category')
```

In [87]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   ProdTaken                 4888 non-null   category
 1   TypeofContact             4888 non-null   category
 2   CityTier                  4888 non-null   category
 3   DurationOfPitch           4888 non-null   float64 
 4   Occupation                4888 non-null   category
 5   Gender                    4888 non-null   object  
 6   NumberOfPersonVisiting    4888 non-null   float64 
 7   NumberOfFollowups         4888 non-null   float64 
 8   ProductPitched            4888 non-null   category
 9   PreferredPropertyStar     4888 non-null   category
 10  MaritalStatus             4888 non-null   category
 11  NumberOfTrips             4888 non-null   float64 
 12  Passport                  4888 non-null   category
 13  PitchSatisfactionScore    4888 non-null   category
 14  OwnCar                    4888 non-null   category
 15  NumberOfChildrenVisiting  4888 non-null   float64 
 16  Designation               4888 non-null   category
 17  MonthlyIncome             4888 non-null   float64 
 18  AgeGroup                  4888 non-null   float64 
```

```
dtypes: category(11), float64(7), object(1)
memory usage: 359.7+ KB
```

In [ ]:

Model building - Bagging

Split the Data

- Because theis a significant imbalance in the distribution of the target classes (18% success on ProdTaken vs 82%), we will use stratified sampling to ensure that relative class frequencies are approximately preserved in train and test sets.
- We will be using the stratify parameter in the train_test_split function.

In [88]:

```
#X = creditData.drop("default" , axis=1)
#y = creditData.pop("default")

X = data.drop(['ProdTaken'],axis=1)
X = pd.get_dummies(X,drop_first=True)
#y = data['ProdTaken'].apply(lambda x : 1 if x=='Yes' else 0)
y = data.pop('ProdTaken')
```

In [89]:

```
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=

# Splitting data into training and test set:
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3, random_state=1,
print(X_train.shape, X_test.shape)
```

```
(3421, 34) (1467, 34)
```

Create Functions

In [90]:

```
##  Function to calculate recall score
def get_recall_score(model,flag=True):
    '''
    model : classifier to predict values of X

    '''
    a = [] # defining an empty list to store train and test results
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)
    train_recall = metrics.recall_score(y_train,pred_train)
    test_recall = metrics.recall_score(y_test,pred_test)
    a.append(train_recall) # adding train recall to list
    a.append(test_recall) # adding test recall to list
    if flag == True: # If the flag is set to True then only the following print stateme
        print("Recall on training set : ",metrics.recall_score(y_train,pred_train))
        print("Recall on test set : ",metrics.recall_score(y_test,pred_test))

    return a # returning the list with train and test scores
```

In [91]:

```
##  Function to calculate precision score
def get_precision_score(model,flag=True):
    '''
    model : classifier to predict values of X
```

```
    '''
    b = []  # defining an empty list to store train and test results
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)
    train_precision = metrics.precision_score(y_train,pred_train)
    test_precision = metrics.precision_score(y_test,pred_test)
    b.append(train_precision) # adding train precision to list
    b.append(test_precision) # adding test precision to list
    if flag == True: # If the flag is set to True then only the following print stateme
        print("Precision on training set : ",metrics.precision_score(y_train,pred_train
        print("Precision on test set : ",metrics.precision_score(y_test,pred_test))

    return b # returning the list with train and test scores
```

In [92]:
```
##  Function to calculate accuracy score
def get_accuracy_score(model,flag=True):
    '''
    model : classifier to predict values of X

    '''
    c = [] # defining an empty list to store train and test results
    train_acc = model.score(X_train,y_train)
    test_acc = model.score(X_test,y_test)
    c.append(train_acc) # adding train accuracy to list
    c.append(test_acc) # adding test accuracy to list
    if flag == True: # If the flag is set to True then only the following print stateme
        print("Accuracy on training set : ",model.score(X_train,y_train))
        print("Accuracy on test set : ",model.score(X_test,y_test))

    return c # returning the list with train and test scores
```

In [93]:
```
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                  columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Build Decision Tree Model

- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.

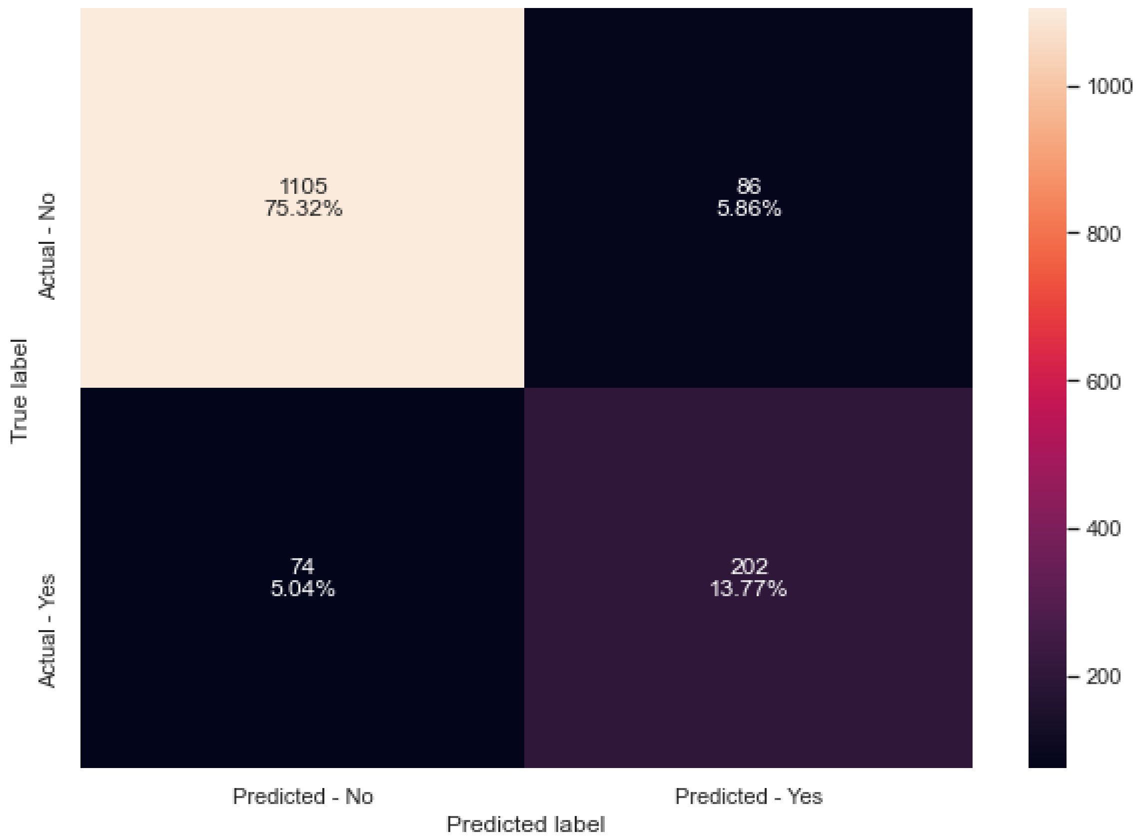
- We will pass a dictionary {0:0.18,1:0.82} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.

In [94]:
```
dtree = DecisionTreeClassifier(criterion='gini',class_weight={0:0.18,1:0.82},random_sta
```

In [95]:
```
dtree.fit(X_train, y_train)
```

Out[95]:
```
DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.82}, random_state=1)
```

In [96]:
```
make_confusion_matrix(dtree,y_test)
```



Confusion Matrix -

- Consumer took the product and the model predicted it correctly that ProdTaken=1 : True Positive (observed=1,predicted=1)
- Consumer didn't take the product and the model predicted ProdTaken=1 : False Positive (observed=0,predicted=1)
- Consumer didn't take the product and the model predicted ProdTaken=0 : True Negative (observed=0,predicted=0)
- Consumer took the product and the model predicted that ProdTaken=0 : False Negative (observed=1,predicted=0)

In [97]:
```
dtree_acc = get_accuracy_score(dtree)
dtree_recall = get_recall_score(dtree)
dtree_precision = get_precision_score(dtree)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.89093387866394
Recall on training set :  1.0
```

```
Recall on test set :  0.7318840579710145
Precision on training set :  1.0
Precision on test set :  0.7013888888888888
```

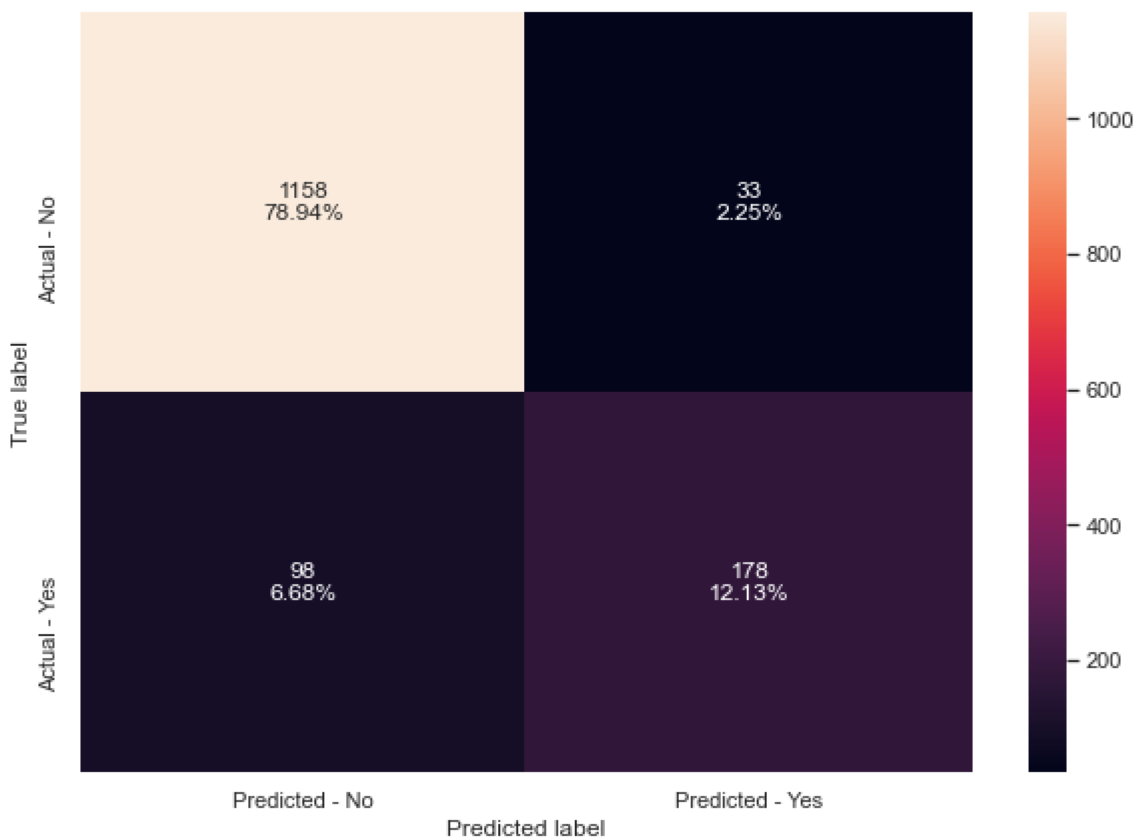
- Decision Treemodel is overfitting on the training set and is performing poorly on the test set in terms of recall.

Build Bagging Classifier Model

In [98]:
```
bagging = BaggingClassifier(random_state=1)
bagging.fit(X_train,y_train)
```

Out[98]:
```
BaggingClassifier(random_state=1)
```

In [99]:
```
make_confusion_matrix(bagging,y_test)
```



In [100…
```
bagging_acc = get_accuracy_score(bagging)
bagging_recall = get_recall_score(bagging)
bagging_precision = get_precision_score(bagging)
```

```
Accuracy on training set :  0.9944460684010523
Accuracy on test set :  0.9107021131561008
Recall on training set :  0.9720496894409938
Recall on test set :  0.644927536231884
Precision on training set :  0.9984051036682615
Precision on test set :  0.8436018957345972
```

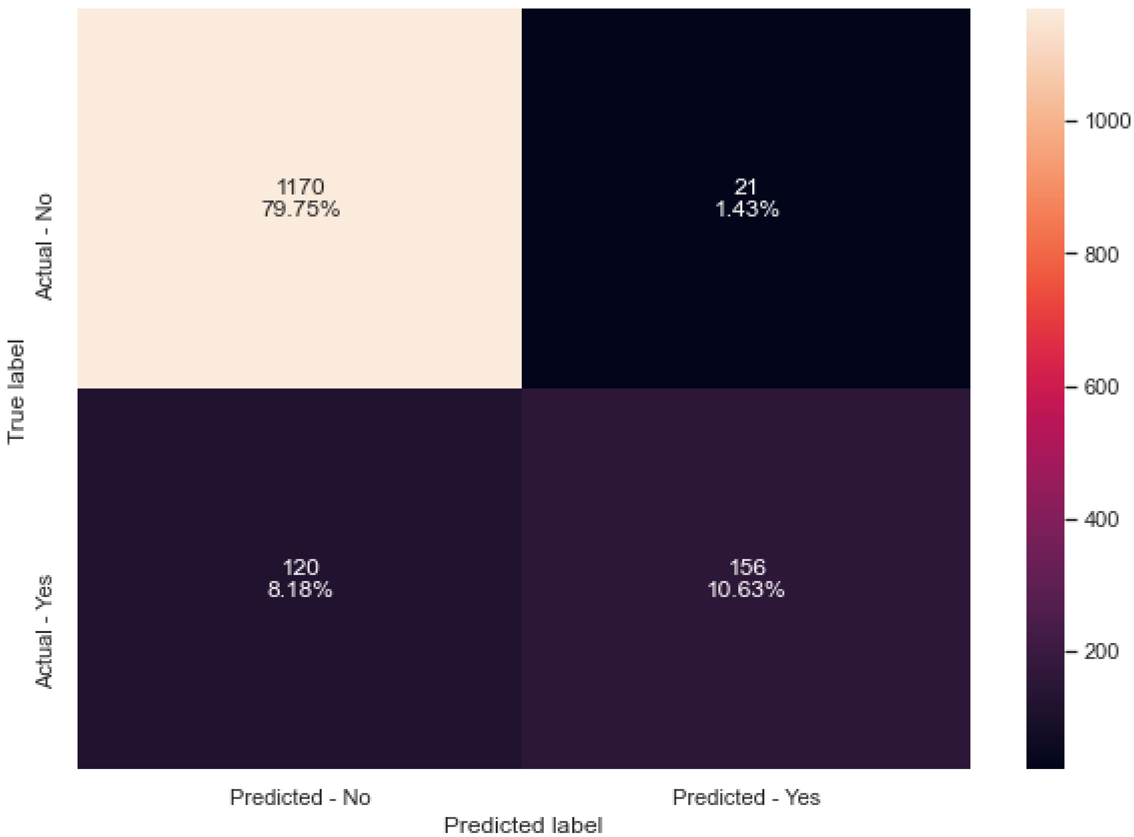
- Bagging classifier is overfitting on the training set and is performing poorly on the test set in terms of recall.

Bagging Classifier with weighted decision tree

In [101…
```
bagging_wt = BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='gini',c
bagging_wt.fit(X_train,y_train)
```

Out[101…
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight={0: 0.18,
                                                                      1: 0.82},
                                                        random_state=1),
                  random_state=1)
```

In [102…
```
make_confusion_matrix(bagging_wt,y_test)
```



In [103…
```
wt_bagging_acc = get_accuracy_score(bagging_wt)
wt_bagging_recall = get_recall_score(bagging_wt)
wt_bagging_precision = get_precision_score(bagging_wt)
```

```
Accuracy on training set :  0.9956153171587255
Accuracy on test set :  0.9038854805725971
Recall on training set :  0.9782608695652174
Recall on test set :  0.5652173913043478
Precision on training set :  0.9984152139461173
Precision on test set :  0.8813559322033898
```

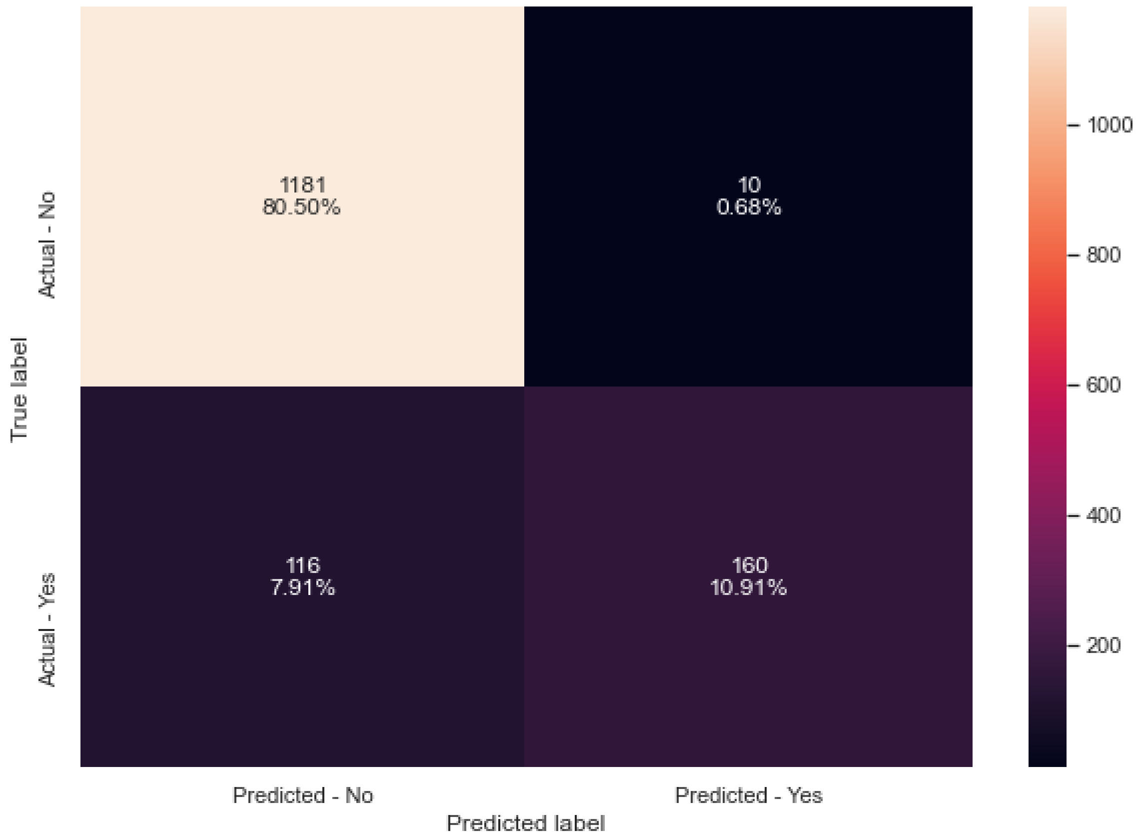
- Bagging classifier with a weighted decision tree is giving very good accuracy and prediction but is not able to generalize well on test data in terms of recall.

Build Random Forest Model

In [104…
```
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train,y_train)
```

Out[104... RandomForestClassifier(random_state=1)

In [105...
```
make_confusion_matrix(rf,y_test)
```



In [106...
```
rf_acc = get_accuracy_score(rf)
rf_recall = get_recall_score(rf)
rf_precision = get_precision_score(rf)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.9141104294478528
Recall on training set :  1.0
Recall on test set :  0.5797101449275363
Precision on training set :  1.0
Precision on test set :  0.9411764705882353
```

- Random Forest has performed well in terms of accuracy and precision, but it is not able to generalize well on the test data in terms of recall.

Random forest with class weights

In [107...
```
rf_wt = RandomForestClassifier(class_weight={0:0.18,1:0.82}, random_state=1)
rf_wt.fit(X_train,y_train)
```

Out[107... RandomForestClassifier(class_weight={0: 0.18, 1: 0.82}, random_state=1)

In [108...
```
make_confusion_matrix(rf_wt,y_test)
```

In [109…

```
wt_rf_acc = get_accuracy_score(rf_wt)
wt_rf_recall = get_recall_score(rf_wt)
wt_rf_precision = get_precision_score(rf_wt)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.901840490797546
Recall on training set :  1.0
Recall on test set :  0.5253623188405797
Precision on training set :  1.0
Precision on test set :  0.9177215189873418
```

- There is not much improvement in metrics of weighted random forest as compared to the unweighted random forest.
- Random Forest with Weighted Tree has performed well in terms of accuracy and precision, but it is not able to generalize well on the test data in terms of recall.

Tuning Models

Using GridSearch for Hyperparameter tuning model

- Grid search is a tuning technique that attempts to compute the optimum values of hyperparameters.
- It is an exhaustive search that is performed on a the specific parameter values of a model.

Tuning Decision Tree

In [110…

```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.18,1:0.82},random_state=1)
```

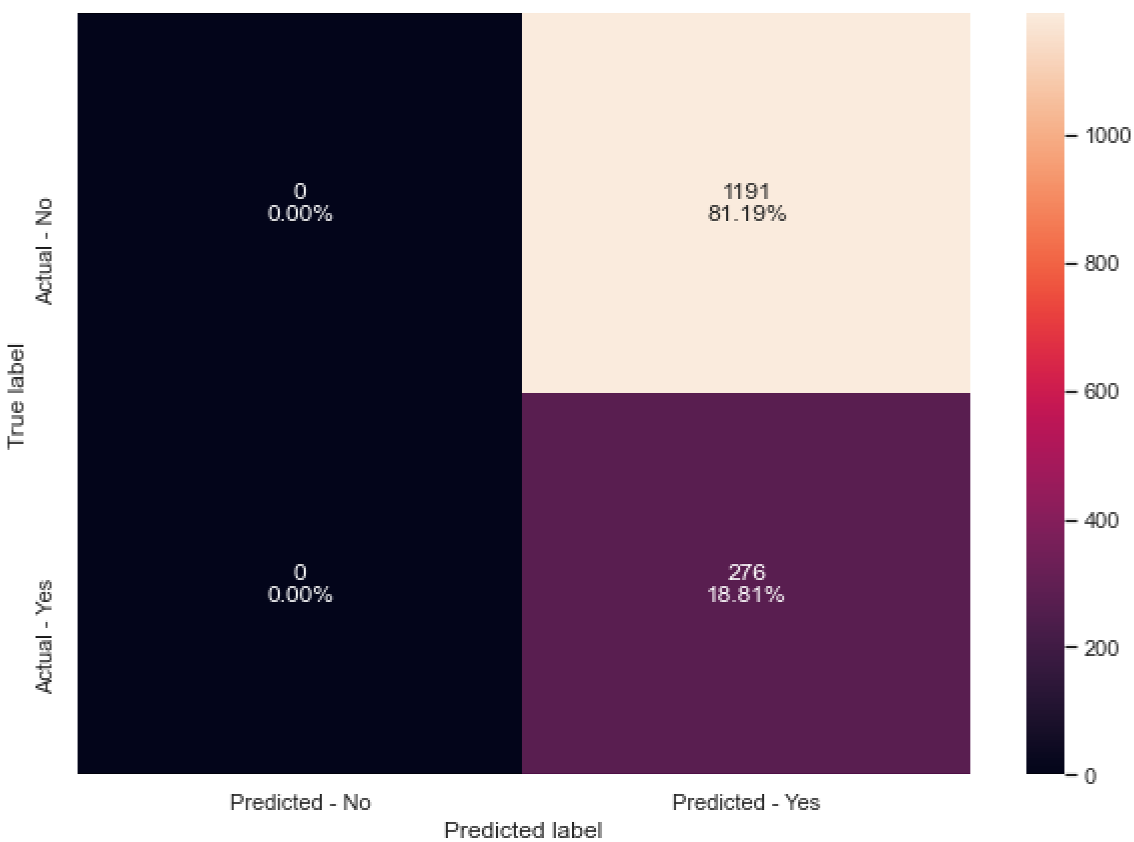
```
# Grid of parameters to choose from - this will perform 2800 tests
parameters = {'max_depth': np.arange(2,30),
              'min_samples_leaf': [1, 2, 5, 7, 10],
              'max_leaf_nodes' : [2, 3, 5, 10,15],
              'min_impurity_decrease': [0.0001,0.001,0.01,0.1]
             }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_estimator.fit(X_train, y_train)
```

Out[110…

```
DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.82}, max_depth=2,
                       max_leaf_nodes=2, min_impurity_decrease=0.1,
                       random_state=1)
```

In [111…

```
make_confusion_matrix(dtree_estimator,y_test)
```



In [112…

```
tuned_dtree_acc = get_accuracy_score(dtree_estimator)
tuned_dtree_recall = get_recall_score(dtree_estimator)
tuned_dtree_precision = get_precision_score(dtree_estimator)
```

```
Accuracy on training set :  0.1882490499853844
Accuracy on test set :  0.18813905930470348
Recall on training set :  1.0
Recall on test set :  1.0
```

```
Precision on training set :  0.18824904998538844
Precision on test set :  0.18813905930470348
```

- Overfitting in decision tree has reduced Accuracy and Precision, but the Recall has improved. This is an indication that overall the model is making many mistakes.

Tuning Bagging Classifier

In [113…

```
# grid search for bagging classifier
cl1 = DecisionTreeClassifier(class_weight={0:0.18,1:0.82},random_state=1)
param_grid = {'base_estimator':[cl1],
              'n_estimators':[5,7,15,51,101],
              'max_features': [0.7,0.8,0.9,1]
             }

grid = GridSearchCV(BaggingClassifier(random_state=1,bootstrap=True), param_grid=param_
grid.fit(X_train, y_train)
```

Out[113…

```
GridSearchCV(cv=5, estimator=BaggingClassifier(random_state=1),
             param_grid={'base_estimator': [DecisionTreeClassifier(class_weight={0: 0.1
8,
                                                                                  1: 0.8
2},
                                                                    random_state=1)],
                         'max_features': [0.7, 0.8, 0.9, 1],
                         'n_estimators': [5, 7, 15, 51, 101]},
             scoring='recall')
```

In [114…

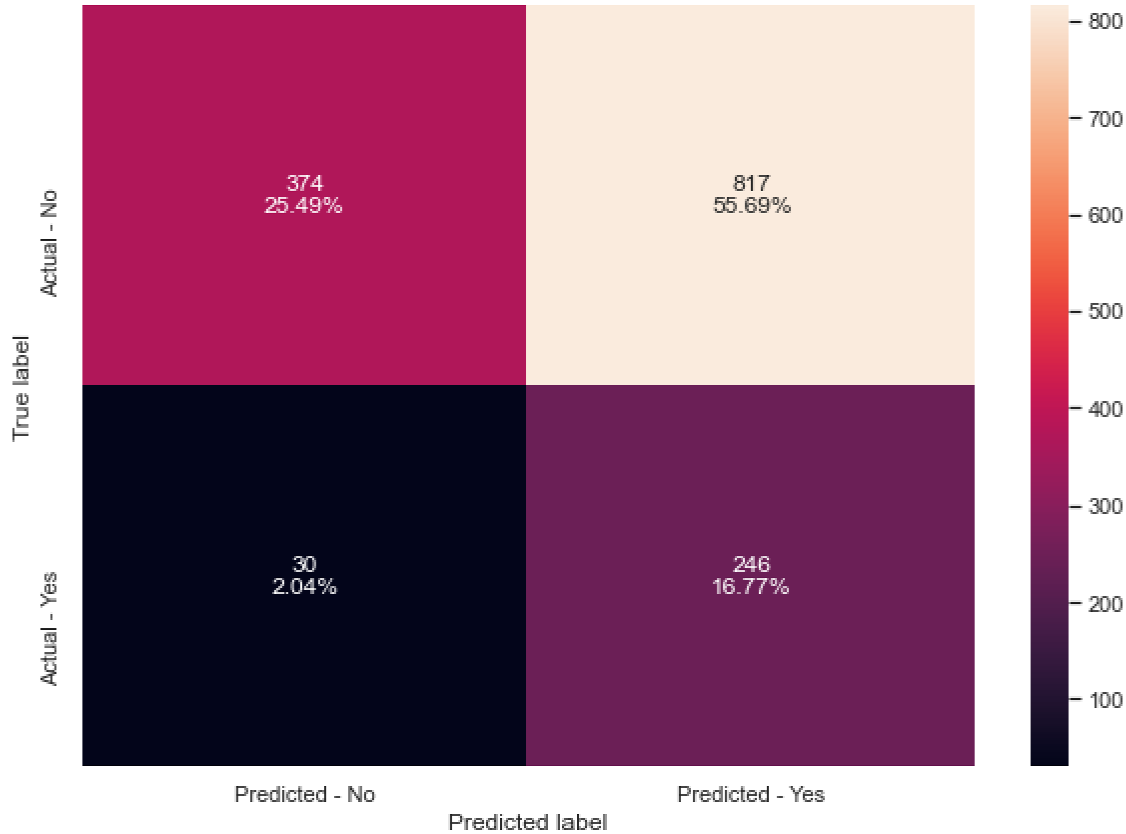
```
## getting the best estimator
bagging_estimator  = grid.best_estimator_
bagging_estimator.fit(X_train,y_train)
```

Out[114…

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight={0: 0.18,
                                                                      1: 0.82},
                                                        random_state=1),
                  max_features=1, n_estimators=51, random_state=1)
```

In [115…

```
make_confusion_matrix(bagging_estimator,y_test)
```



In [116…

```
tuned_bagging_acc= get_accuracy_score(bagging_estimator)
tuned_bagging_recall = get_recall_score(bagging_estimator)
tuned_bagging_precision = get_precision_score(bagging_estimator)
```

```
Accuracy on training set :  0.4811458637825197
Accuracy on test set :  0.4226312201772324
Recall on training set :  0.9611801242236024
Recall on test set :  0.8913043478260869
Precision on training set :  0.2612916842549599
Precision on test set :  0.23142050799623706
```

- Recall has improved but the accuracy and precision of the model has dropped drastically which is an indication that overall the model is making many mistakes.

Tuning Random Forest

In [117…

```
# Choose the type of classifier.
rf_estimator = RandomForestClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
        "n_estimators": [110,251],
        "min_samples_leaf": np.arange(1, 6,1),
        "max_features": [0.7,0.9,'log2','auto'],
        "max_samples": [0.7,0.9,None],
}


# Run the grid search
grid_obj = GridSearchCV(rf_estimator, parameters, scoring='recall',cv=5)
grid_obj = grid_obj.fit(X_train, y_train)
```

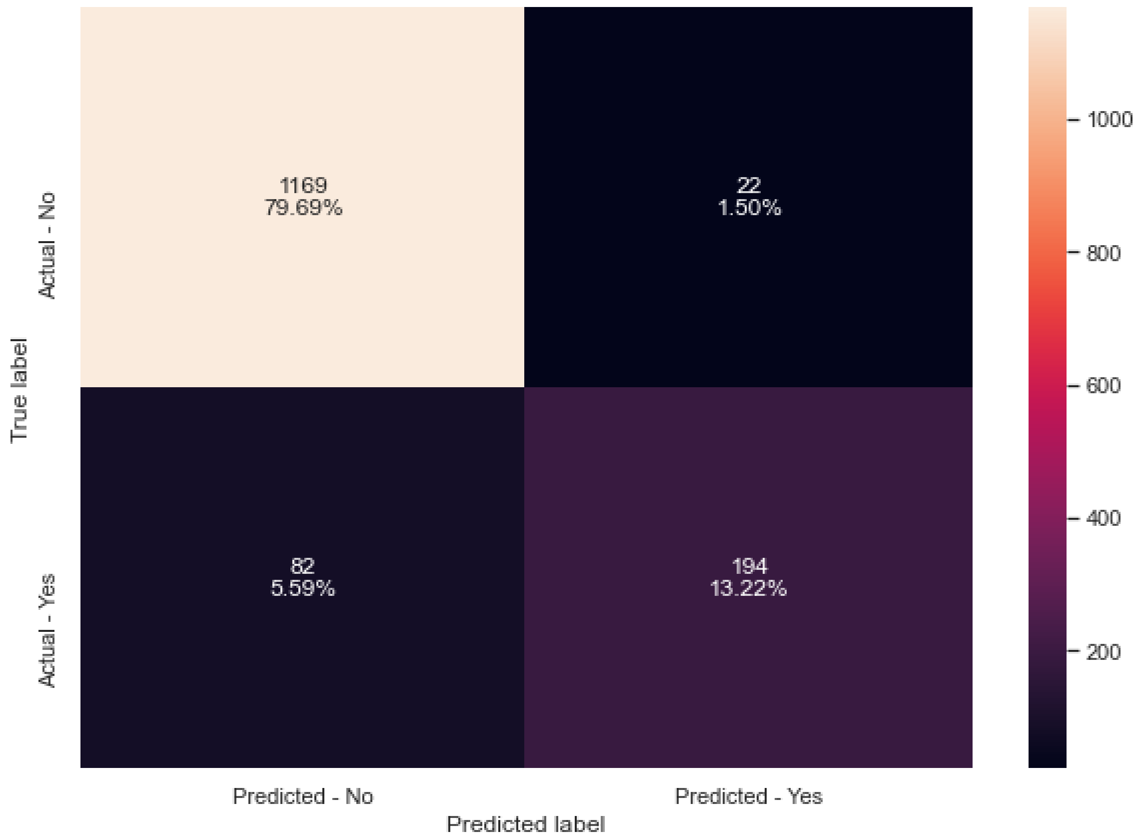
```
# Set the clf to the best combination of parameters
rf_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_estimator.fit(X_train, y_train)
```

Out[117…

```
RandomForestClassifier(max_features=0.7, n_estimators=251, random_state=1)
```

In [118…

```
make_confusion_matrix(rf_estimator,y_test)
```



In [119…

```
tuned_rf_acc = get_accuracy_score(rf_estimator)
tuned_rf_recall = get_recall_score(rf_estimator)
tuned_rf_precision = get_precision_score(rf_estimator)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.929107021131561
Recall on training set :  1.0
Recall on test set :  0.7028985507246377
Precision on training set :  1.0
Precision on test set :  0.8981481481481481
```

- Recall has improved and the accuracy and precision of the model has improved drastically.

Comparing all the models

In [122…

```
# defining list of models
models = [dtree,dtree_estimator,bagging,bagging_wt,bagging_estimator,rf,rf_wt,rf_estima
# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
```

```python
recall_test = []
precision_train = []
precision_test = []

# Looping through all the models to get the accuracy,recall and precision scores
for model in models:
    # accuracy score
    j = get_accuracy_score(model,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    # recall score
    k = get_recall_score(model,False)
    recall_train.append(k[0])
    recall_test.append(k[1])
    # precision score
    l = get_precision_score(model,False)
    precision_train.append(l[0])
    precision_test.append(l[1])
```

In [123…

```python
comparison_frame = pd.DataFrame({'Model':['Decision Tree','Tuned Decision Tree','Baggin
                                          'Weighted Bagging Classifier','Tuned Bagging
                                          'Random Forest','Weighted Random Forest','Tun
                                          'Train_Accuracy': acc_train,
                                          'Test_Accuracy': acc_test,
                                          'Train_Recall': recall_train,
                                          'Test_Recall': recall_test,
                                          'Train_Precision': precision_train,
                                          'Test_Precision': precision_test})

comparison_frame
```

Out[123…

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision
0	Decision Tree	1.000000	0.890934	1.000000	0.731884	1.000000	0.701389
1	Tuned Decision Tree	0.188249	0.188139	1.000000	1.000000	0.188249	0.188139
2	Bagging Classifier	0.994446	0.910702	0.972050	0.644928	0.998405	0.843602
3	Weighted Bagging Classifier	0.995615	0.903885	0.978261	0.565217	0.998415	0.881356
4	Tuned Bagging Classifier	0.481146	0.422631	0.961180	0.891304	0.261292	0.231421
5	Random Forest	1.000000	0.914110	1.000000	0.579710	1.000000	0.941176
6	Weighted Random Forest	1.000000	0.901840	1.000000	0.525362	1.000000	0.917722
7	Tuned Random Forest	1.000000	0.929107	1.000000	0.702899	1.000000	0.898148

- Decision tree performed well on training and test set.
- Bagging classifier overfitted the data before and after tuning.
- Random Forest with default parameters performed better after tuning.

Feature importance of Random Forest with Tuning

In [124…

```
# importance of features in the tree building ( The importance of a feature is computed
#(normalized) total reduction of the criterion brought by that feature. It is also know

print (pd.DataFrame(rf_estimator.feature_importances_, columns = ["Imp"], index = X_tra
```

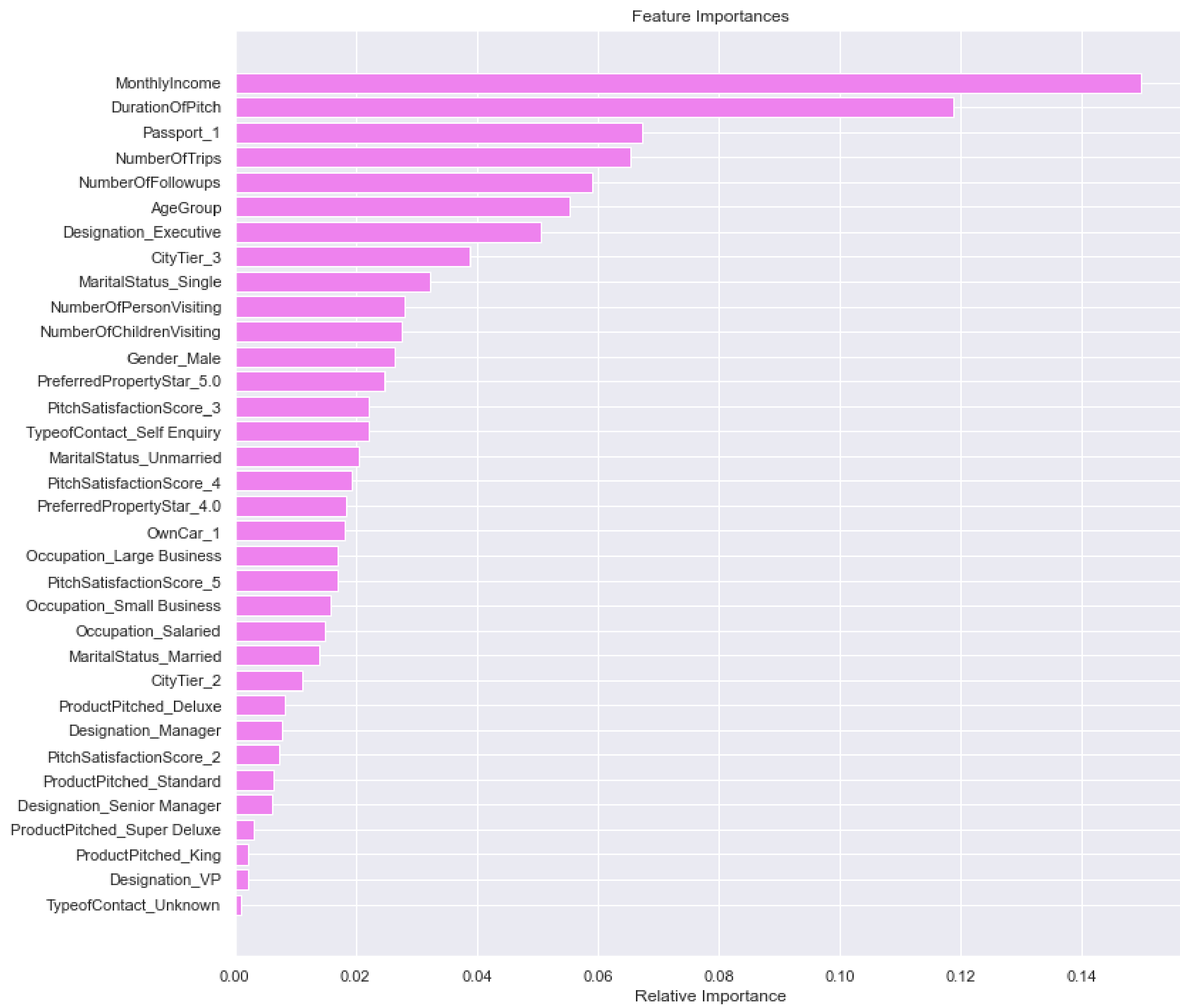
```
                                   Imp
MonthlyIncome                 0.149993
DurationOfPitch               0.118809
Passport_1                    0.067302
NumberOfTrips                 0.065421
NumberOfFollowups             0.059158
AgeGroup                      0.055359
Designation_Executive         0.050722
CityTier_3                    0.038826
MaritalStatus_Single          0.032236
NumberOfPersonVisiting        0.028049
NumberOfChildrenVisiting      0.027526
Gender_Male                   0.026446
PreferredPropertyStar_5.0     0.024879
PitchSatisfactionScore_3      0.022245
TypeofContact_Self Enquiry    0.022170
MaritalStatus_Unmarried       0.020512
PitchSatisfactionScore_4      0.019415
PreferredPropertyStar_4.0     0.018364
OwnCar_1                      0.018127
Occupation_Large Business     0.017021
PitchSatisfactionScore_5      0.016978
Occupation_Small Business     0.015879
Occupation_Salaried           0.014933
MaritalStatus_Married         0.014003
CityTier_2                    0.011154
ProductPitched_Deluxe         0.008305
Designation_Manager           0.007794
PitchSatisfactionScore_2      0.007411
ProductPitched_Standard       0.006327
Designation_Senior Manager    0.006286
ProductPitched_Super Deluxe   0.003037
ProductPitched_King           0.002246
Designation_VP                0.002162
TypeofContact_Unknown         0.000905
```

In [125…

```
feature_names = X_train.columns
```

In [126…

```
importances = rf_estimator.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- `Monthlyincome` is the most important feature for prediction followed by `DurationOfPitch`, `Passport_1` and `NumberOfTrips`.

In [ ]:

```

```

In [ ]:

```

```

Model building - Boosting

In [135…

```
##  Function to calculate r2_score and RMSE on train and test data
def get_model_score(model, flag=True):
    '''
    model : classifier to predict values of X

    '''
    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    train_r2=metrics.r2_score(y_train,pred_train)
```

```
    test_r2=metrics.r2_score(y_test,pred_test)
    train_rmse=np.sqrt(metrics.mean_squared_error(y_train,pred_train))
    test_rmse=np.sqrt(metrics.mean_squared_error(y_test,pred_test))

    #Adding all scores in the list
    score_list.extend((train_r2,test_r2,train_rmse,test_rmse))

    # If the flag is set to True then only the following print statements will be dispa
    if flag==True:
        print("R-sqaure on training set : ",metrics.r2_score(y_train,pred_train))
        print("R-square on test set : ",metrics.r2_score(y_test,pred_test))
        print("RMSE on training set : ",np.sqrt(metrics.mean_squared_error(y_train,pred
        print("RMSE on test set : ",np.sqrt(metrics.mean_squared_error(y_test,pred_test

    # returning the list with train and test scores
    return score_list
```

AdaBoost Regressor Model

In [148...

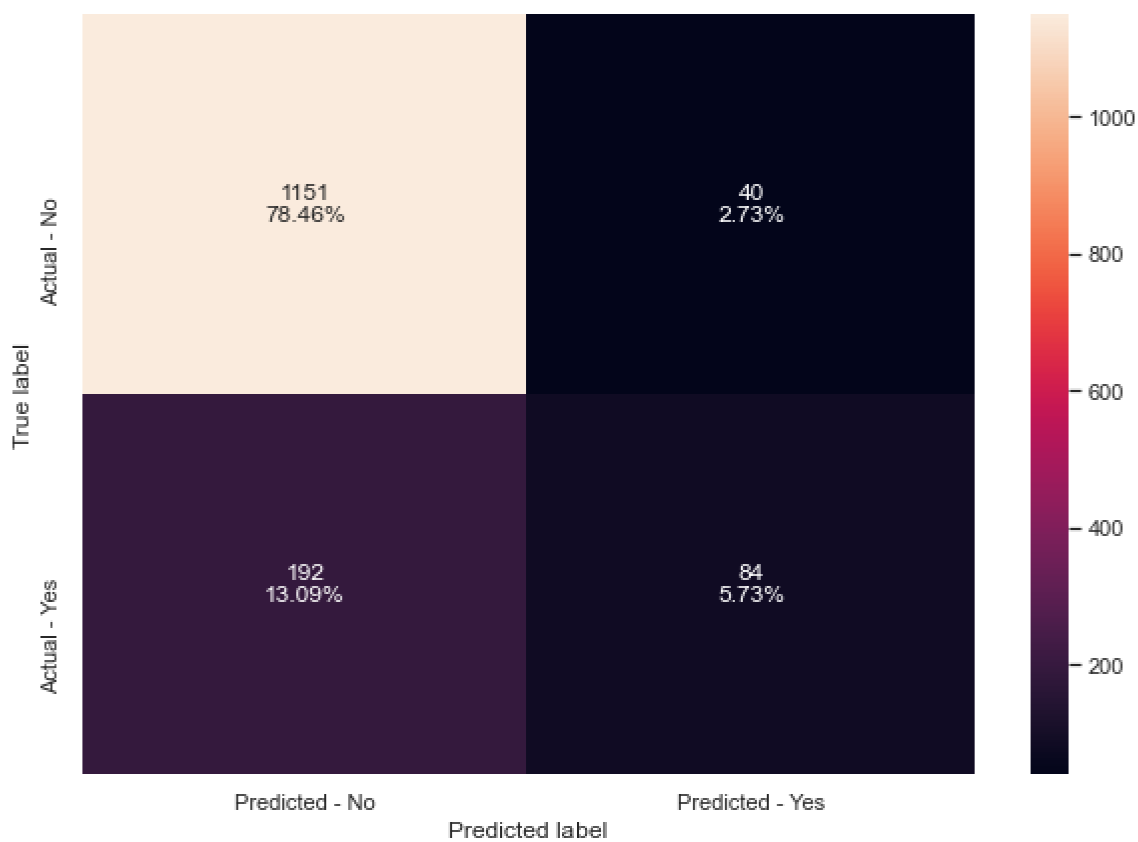
```
abc = AdaBoostClassifier(random_state=1)
abc.fit(X_train,y_train)
```

Out[148...

```
AdaBoostClassifier(random_state=1)
```

In [149...

```
make_confusion_matrix(abc,y_test)
```



In [147...

```
abc_score=get_metrics_score(abc)
```

```
Accuracy on training set :  0.842443729903537
Accuracy on test set :  0.841854124062713
Recall on training set :  0.3059006211180124
Recall on test set :  0.30434782608695654
```

```
Precision on training set :  0.6816608996539792
Precision on test set :  0.6774193548387096
```

- AdaBoost is generalizing well but it is giving very poor performance on recall.

Hyperparameter Tuning

In [150…

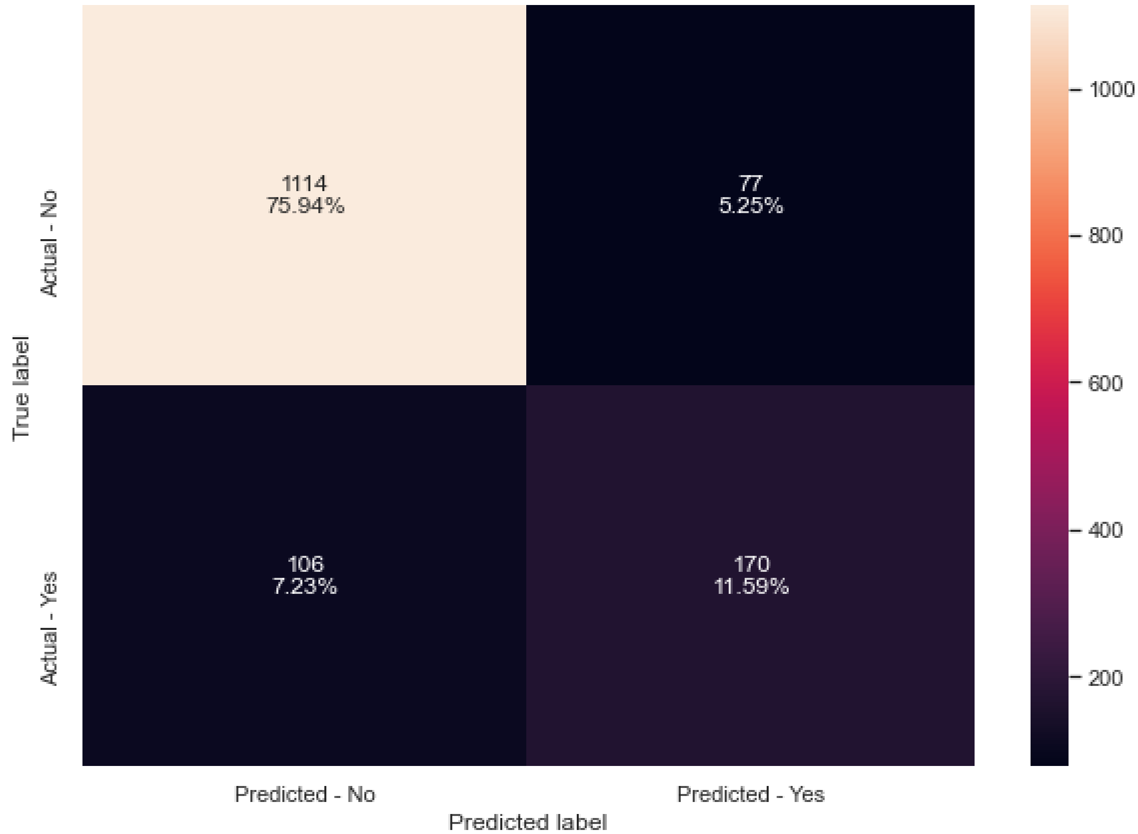
```python
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {
    #Let's try different max_depth for base_estimator
    "base_estimator":[DecisionTreeClassifier(max_depth=1),DecisionTreeClassifier(max_de
    "n_estimators": np.arange(10,110,10),
    "learning_rate":np.arange(0.1,2,0.1)
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(abc_tuned, parameters, scoring=acc_scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
abc_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
abc_tuned.fit(X_train, y_train)
```

Out[150…

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   learning_rate=1.5000000000000002, n_estimators=100,
                   random_state=1)
```

In [151…

```python
make_confusion_matrix(abc_tuned,y_test)
```

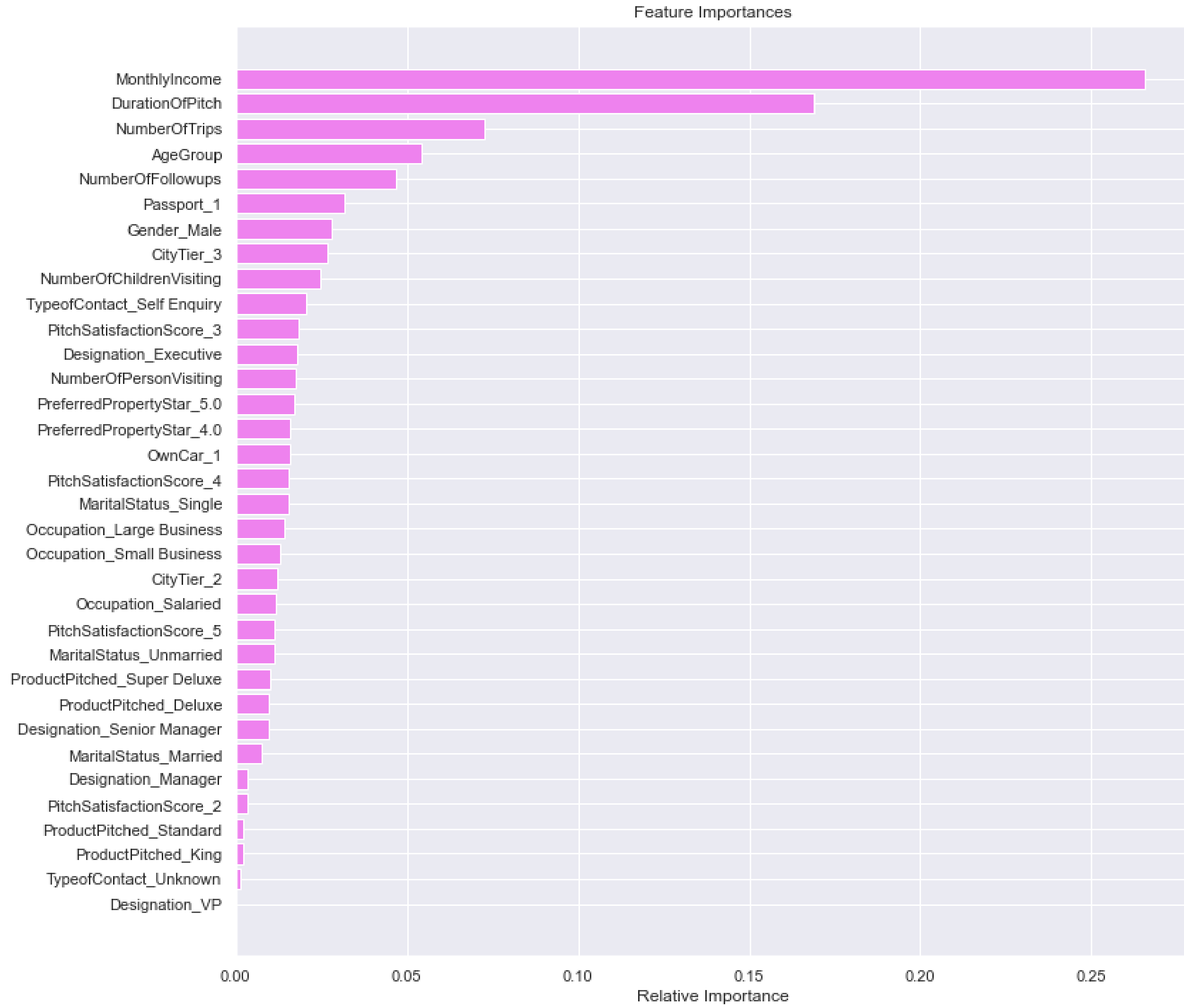


In [152…
```
abc_tuned_score=get_metrics_score(abc_tuned)
```

```
Accuracy on training set :  0.9874305758550131
Accuracy on test set :  0.8752556237218814
Recall on training set :  0.953416149068323
Recall on test set :  0.6159420289855072
Precision on training set :  0.9792663476874003
Precision on test set :  0.6882591093117408
```

- The model is overfitting the train data as train accuracy is much higher than the test accuracy.
- The model has low test recall. This implies that the model is not good at identifying defaulters.

In [153…
```
importances = abc_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



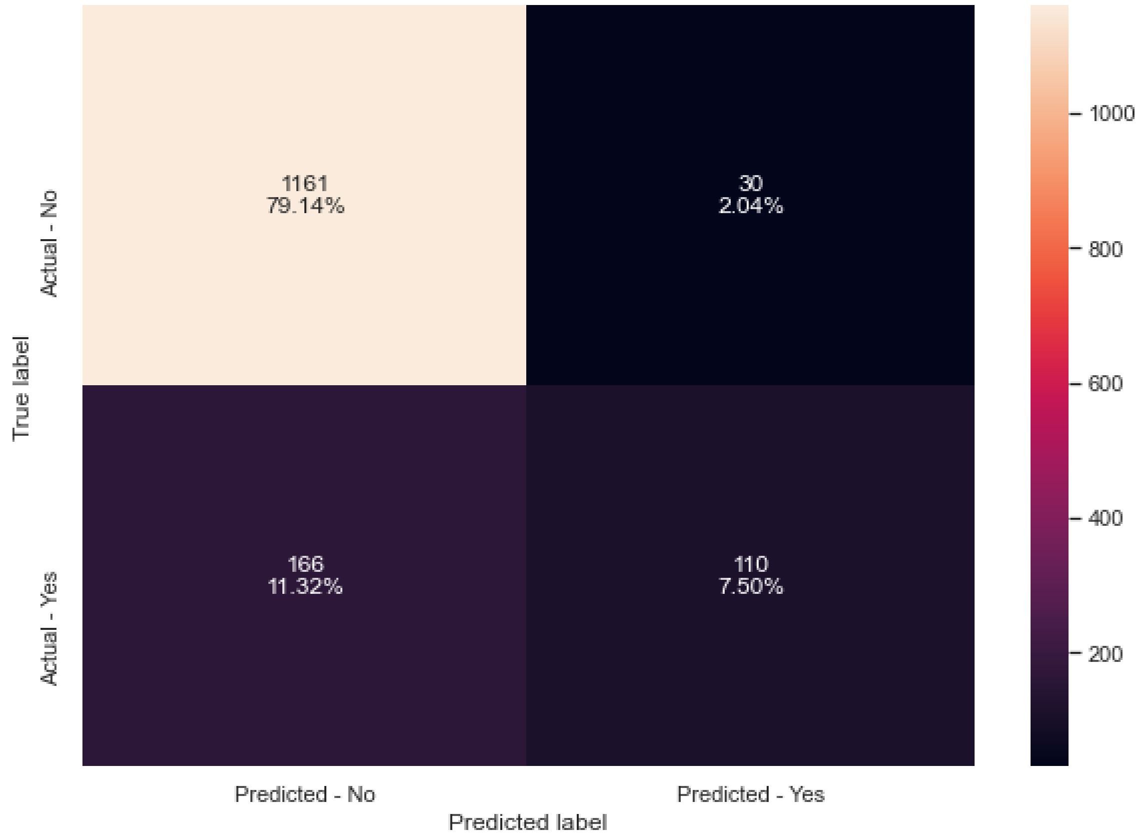
- `MonthlyIncome` is the most important feature as per the tuned AdaBoost model, followed by `DurationOfPitch`, `NumberOfTrips` and `AgeGroup`.

Gradient Boosting Classifier Model

In [154…
```
gbc_init = GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),random_st
gbc_init.fit(X_train,y_train)
```

Out[154…
```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           random_state=1)
```

In [155…
```
make_confusion_matrix(gbc_init,y_test)
```



In [156…

```
#Using above defined function to get accuracy, recall and precision on train and test s
gbc_init_score=get_metrics_score(gbc_init)
```

```
Accuracy on training set :  0.8827828120432623
Accuracy on test set :  0.8663940013633266
Recall on training set :  0.44254658385093165
Recall on test set :  0.39855072463768115
Precision on training set :  0.8715596330275229
Precision on test set :  0.7857142857142857
```

- Gradient boosting is generalizing well and giving poor results on recall.

Hyperparameter Tuning

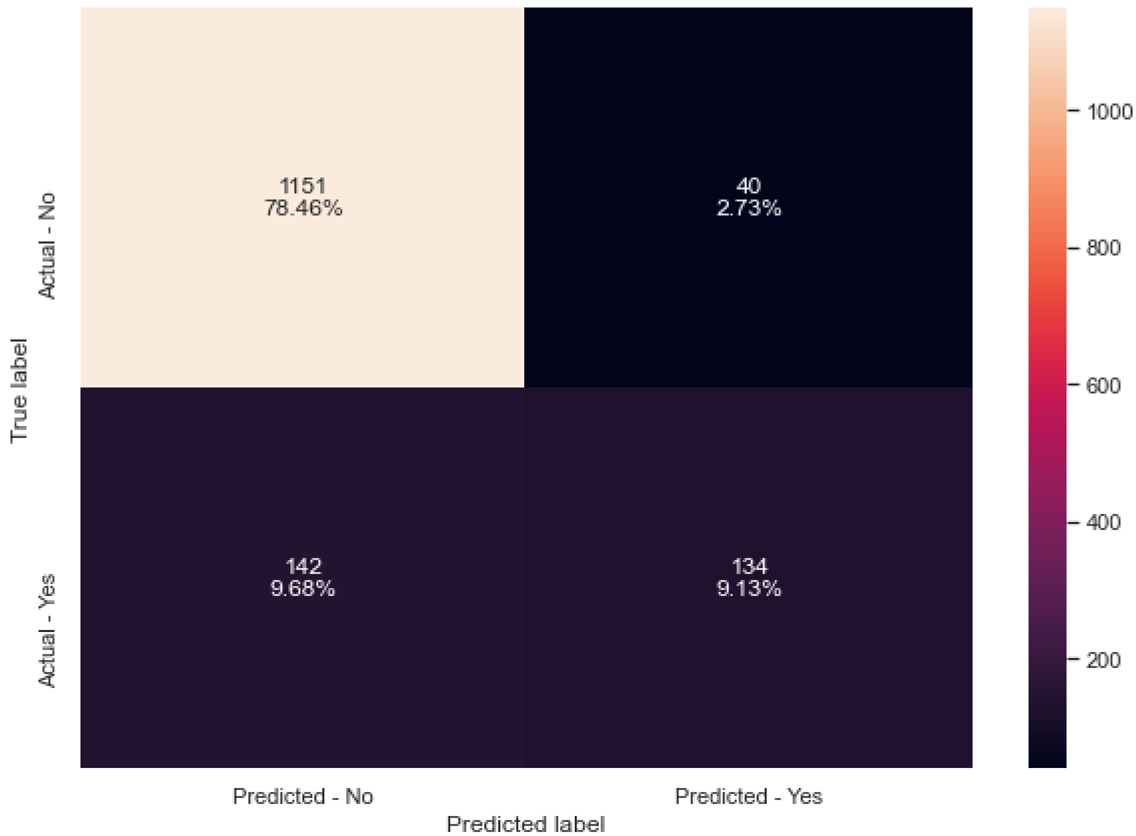
In [157…

```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),random_s

# Grid of parameters to choose from
## add from article
parameters = {
    "n_estimators": [100,150,200],
    "subsample":[0.8,0.9,1],
    "max_features":[0.7,0.8,0.9,1]
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=acc_scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
```

```
gbc_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
gbc_tuned.fit(X_train, y_train)
```

Out[157… GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.9, n_estimators=200, random_state=1,
                           subsample=0.9)

In [159…
```
make_confusion_matrix(gbc_tuned,y_test)
```



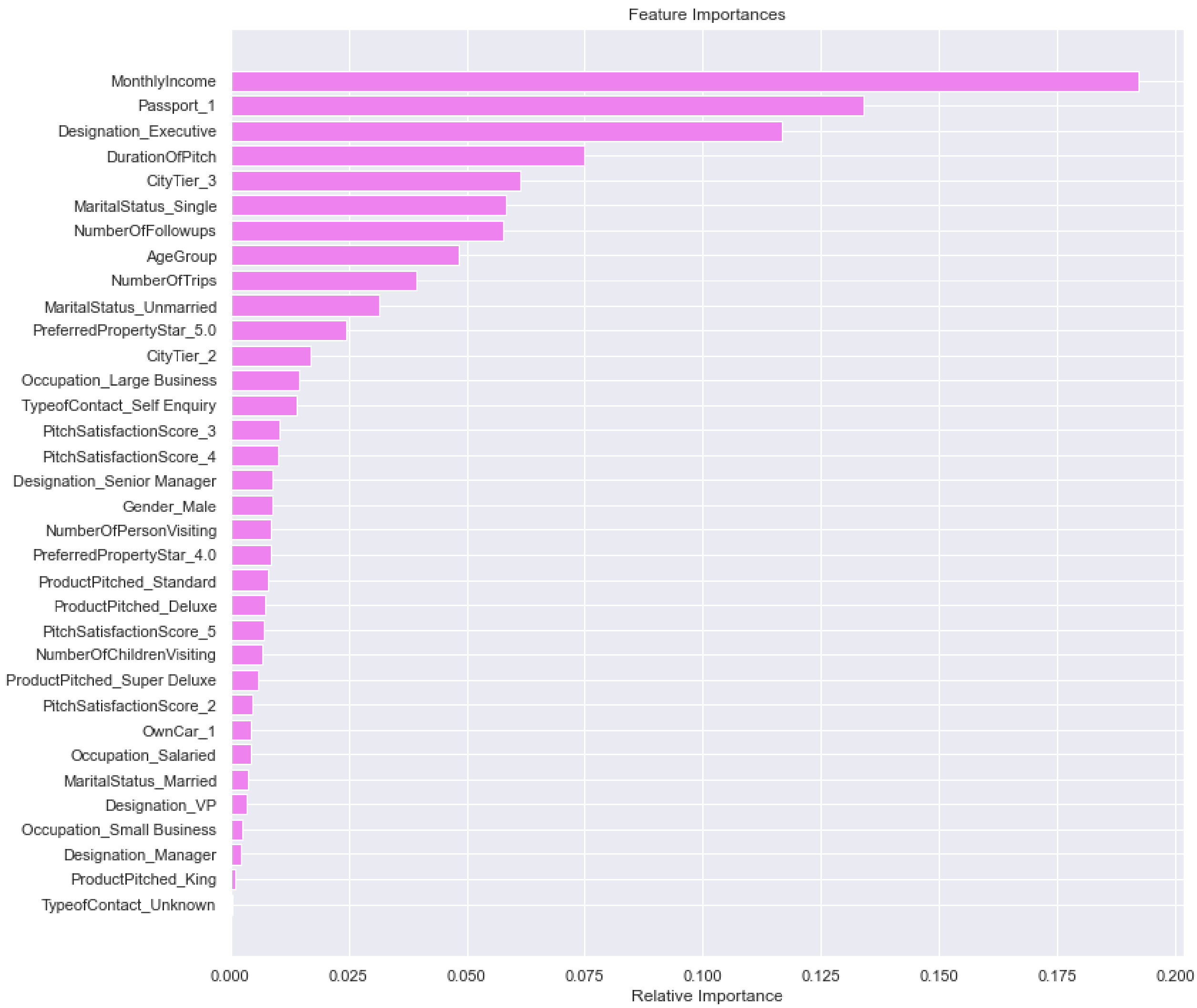
In [160…
```
#Using above defined function to get accuracy, recall and precision on train and test s
gbc_tuned_score=get_metrics_score(gbc_tuned)
```

```
Accuracy on training set :  0.9087985969014908
Accuracy on test set :  0.8759372869802318
Recall on training set :  0.562111801242236
Recall on test set :  0.4855072463768116
Precision on training set :  0.923469387755102
Precision on test set :  0.7701149425287356
```

- The model performace has not increased by much.
- The model has started to overfit the train data in terms of recall.
- The model is generalizing well but it is giving very poor performance on recall.

In [161…
```
importances = gbc_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
```

```
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- `MonthlyIncome` is the most important feature as per the tuned AdaBoost model, followed by `Passport_1`, `Designation_Executive` and `DurationOfPitch`.

XGBoost Classifier Model

In [165…

```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1)

# Grid of parameters to choose from
## add from
parameters = {
    "n_estimators": np.arange(10,100,30),
    "scale_pos_weight":[0,1,2],
    "subsample":[0.5,0.7,1],
    "learning_rate":[0.01,0.1,0.2],
    "gamma":[0,1,3],
    "colsample_bytree":[0.5,0.7,1],
    "colsample_bylevel":[0.5,0.7,1]
}
```

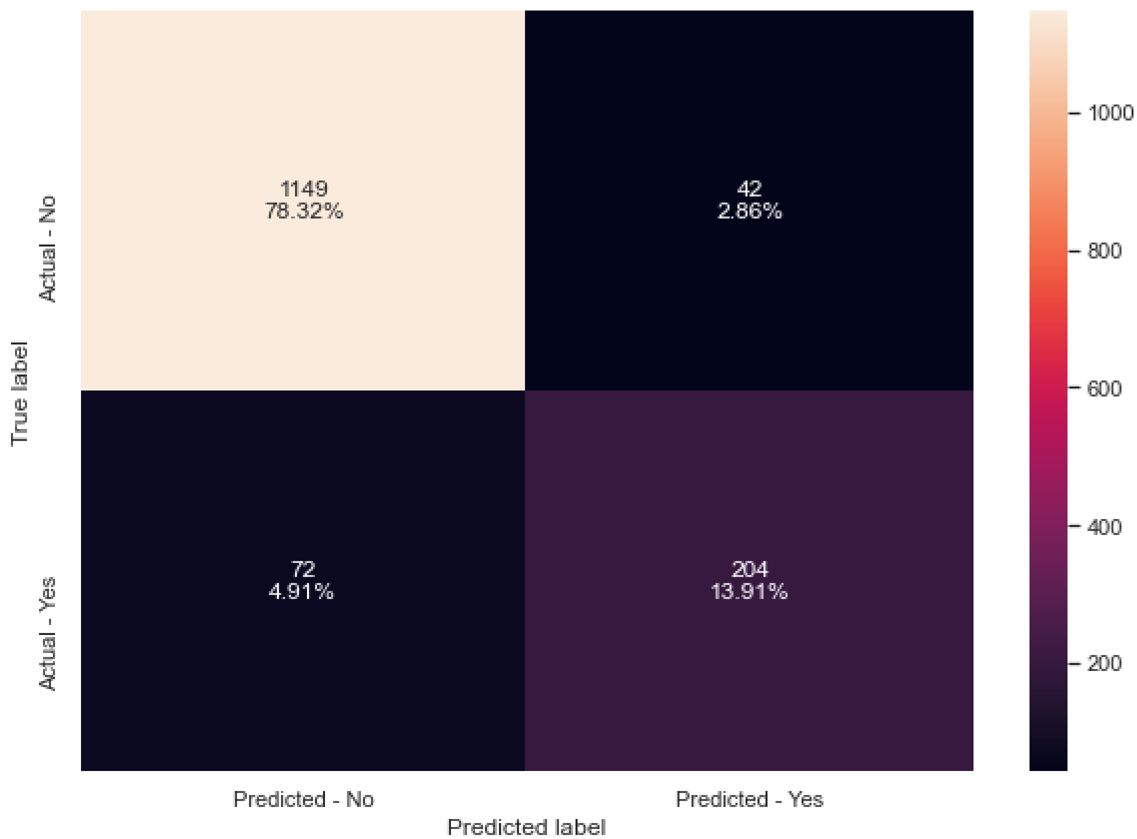
```python
# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters,scoring=acc_scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
xgb_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb_tuned.fit(X_train, y_train)
```

Out[165…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=1, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.2, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=70, n_jobs=4, num_parallel_tree=1, random_state=1,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=2, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [166…

```python
make_confusion_matrix(xgb_tuned,y_test)
```



In [167…

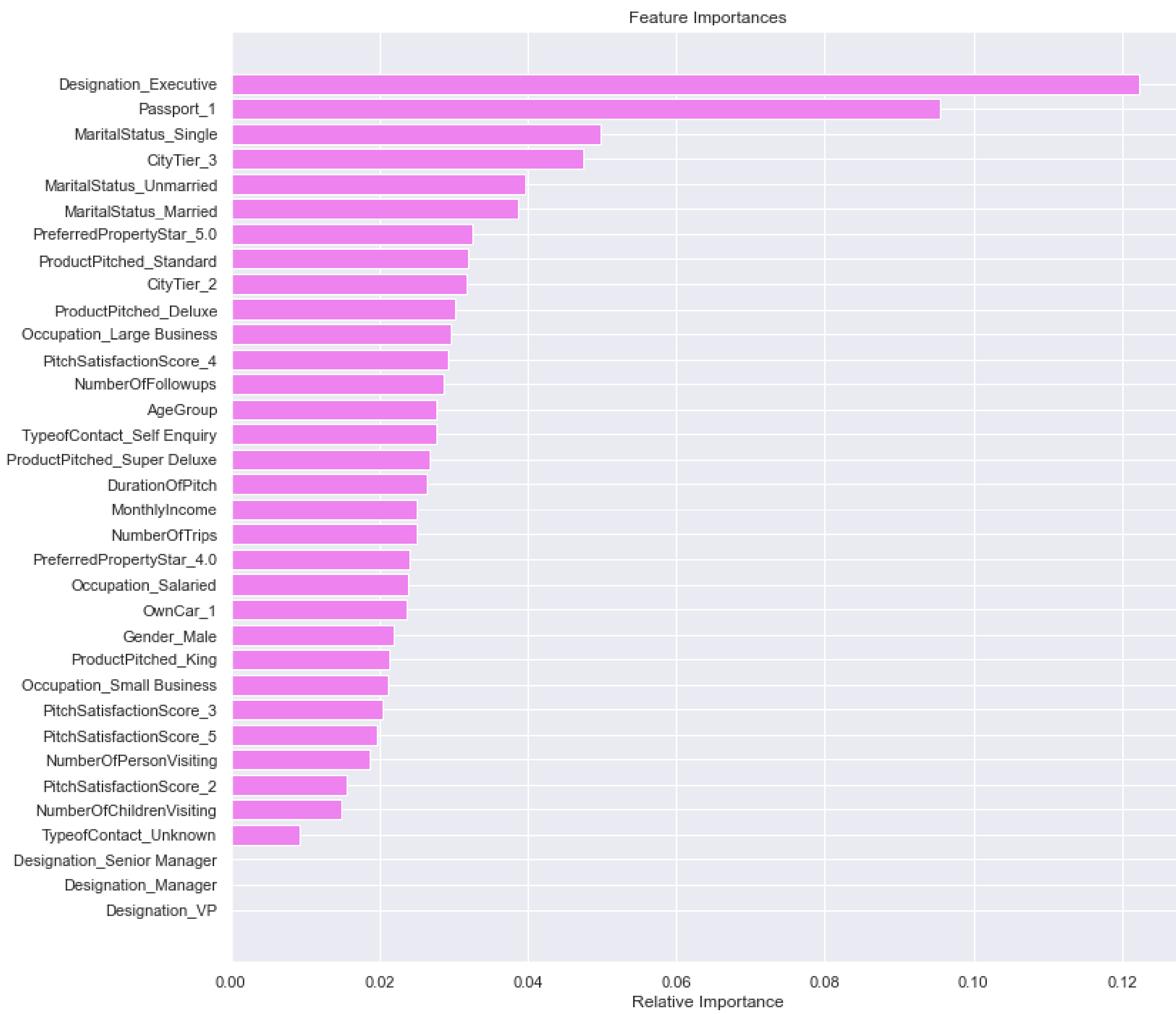
```python
#Using above defined function to get accuracy, recall and precision on train and test
xgb_tuned_score=get_metrics_score(xgb_tuned)
```

```
Accuracy on training set :  0.9929845074539608
Accuracy on test set :  0.9222903885480572
Recall on training set :  0.9782608695652174
Recall on test set :  0.7391304347826086
Precision on training set :  0.984375
Precision on test set :  0.8292682926829268
```

- This model performs really well on all metrics. Even though Recall on test data seems a bit low, it is not that bad and all other metrics look good.

In [168…

```
importances = xgb_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- `Designation_Executive` is the most important feature for prediction followed by `Passport_1`, `MaritalStatus_Single` and `CityTier_3`.

Comparing all models

In [171…

```
# defining list of models
models = [abc, abc_tuned, gbc_init, gbc_tuned, xgb_tuned]

# defining empty lists to add train and test results
acc_train = []
```

```
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []

# looping through all the models to get the accuracy, precall and precision scores
for model in models:
    j = get_metrics_score(model,False)
    acc_train.append(np.round(j[0],2))
    acc_test.append(np.round(j[1],2))
    recall_train.append(np.round(j[2],2))
    recall_test.append(np.round(j[3],2))
    precision_train.append(np.round(j[4],2))
    precision_test.append(np.round(j[5],2))
```

In [173…

```
comparison_frame = pd.DataFrame({'Model':['AdaBoost with default paramters','AdaBoost
                                          'Gradient Boosting with init=AdaBoost','Grad
                                          'XGBoost Tuned'],
                                          'Train_Accuracy': acc_train,'Test_Accuracy':
                                          'Train_Recall':recall_train,'Test_Recall':re
                                          'Train_Precision':precision_train,'Test_Prec

comparison_frame
```

Out[173…

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precisior
0	AdaBoost with default paramters	0.84	0.84	0.31	0.30	0.68	0.68
1	AdaBoost Tuned	0.99	0.88	0.95	0.62	0.98	0.69
2	Gradient Boosting with init=AdaBoost	0.88	0.87	0.44	0.40	0.87	0.79
3	Gradient Boosting Tuned	0.91	0.88	0.56	0.49	0.92	0.77
4	XGBoost Tuned	0.99	0.92	0.98	0.74	0.98	0.83

- Tuned XGBoost model is the best model here. It has really high performance metrics, and consistent recall values.

Business Recommendations

- Company can focus on targeting customer with these strong important features:
  - Designation_Executive
  - Passport_1
  - MaritalStatus_Single
  - CityTier_3

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

End-of-File